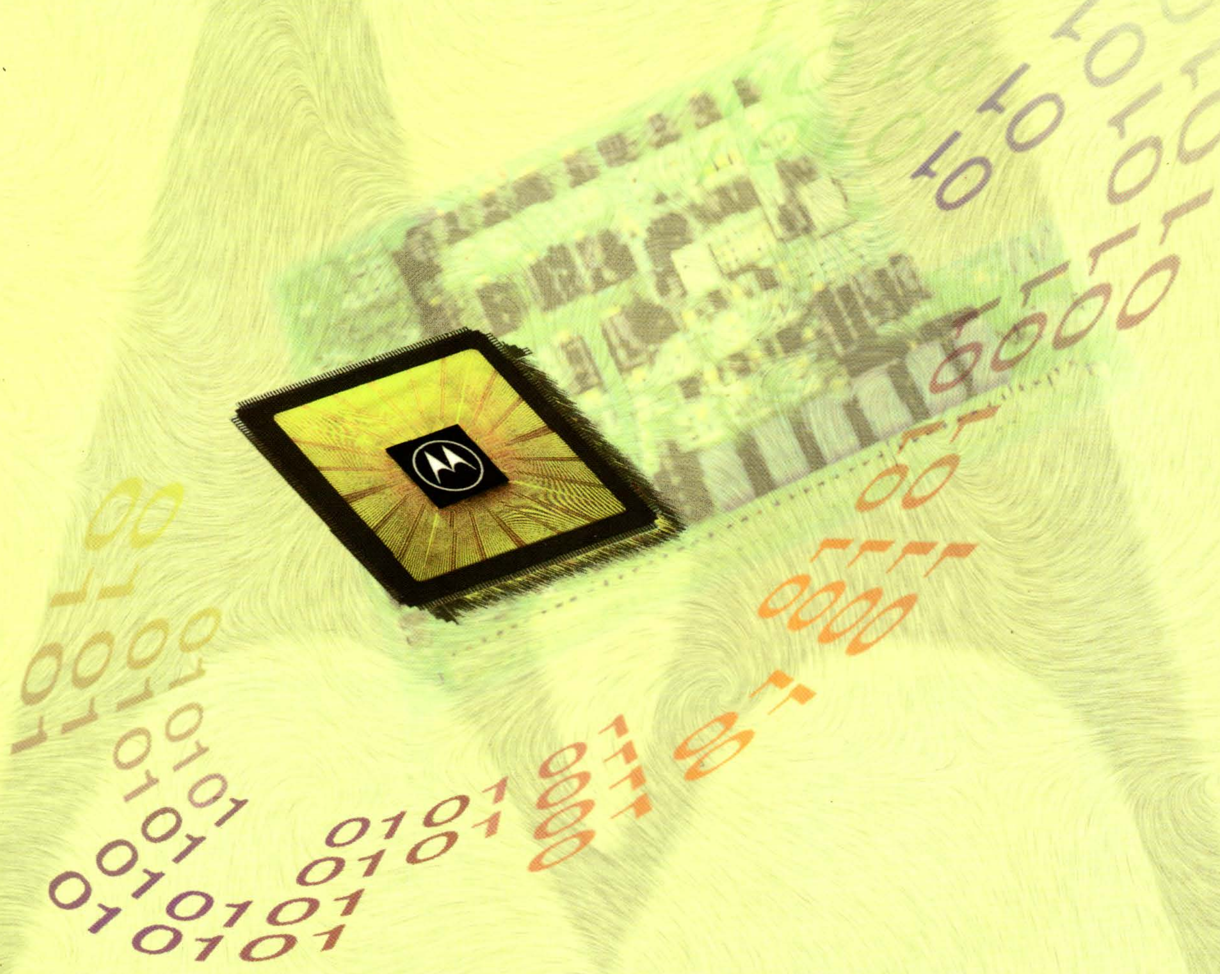
MOTOROLA J,K系列

廉价单片机原理及应用

余永权　林　伟　编



北京航空航天大学出版社

Retro Workshop
果粉工作室

MOTOROLA J,K 系列

廉价单片机原理及应用

余永权　林　伟　编

北京航空航天大学出版社

内 容 简 介

单片机在工业控制、汽车、仪器、家用电器、玩具、计算机外部设备、航空航天、交通运输、通信等领域都有广泛的应用。在众多形式的单片机中，廉价单片机的用量最大。它们主要用于控制任务不复杂、体积小、功能简单、经济性好的产品和场合，尤其在家用电器、玩具、通信产品和某些仪器及设备上的应用是十分有效的。

本书所介绍的MOTOROLA公司J,K系列单片机是该公司结构最简单、引脚最少、价格便宜的单片机。书中介绍了这两个系列单片机的结构原理，并且给出了各种很有实用价值的例子。

本书可供从事单片机应用、智能产品开发的科技人员阅读，也可作为有关单片机控制、家电产品或设备设计人员的参考书，还可作为有关专业学生的参考书。

图书在版编目(CIP)数据

MOTOROLA J,K系列廉价单片机原理及应用/余永权等编著. —北京：北京航空航天大学出版社，1998.11

ISBN 7-81012-764-0

Ⅰ.M… Ⅱ.余… Ⅲ.单片微型计算机-基本知识 Ⅳ.TP368.1

中国版本图书馆CIP数据核字(98)第19357号

MOTOROLA J,K系列廉价单片机原理及应用

余永权 林 伟 编

责任编辑 冯学民

责任校对 张韵秋

北京航空航天大学出版社出版发行

(北京市学院路37号(100083)，发行部电话62015720)

北京宏文印刷厂印装 各地书店经销

开本：787×1092 1/16 印张：14.25 字数：361千字

1998年12月第1版 1998年12月第1次印刷 印数：4 000册

ISBN 7-81012-764-0/TP·272 定价：19.50元

前　言

在产品、特别是智能产品设计中，通常把经济效益放在十分重要的位置，这就需要设计人员在开发和设计中不断提高单片机应用的水平。充分利用单片机的内部资源，在达到同一技术指标的前提下选用简单、廉价的单片机，无论从经济性亦或从科学性上来讲，都是一种明智的选择；另外，也反映了研究设计人员的学术水准和综合能力。

廉价单片机往往是一种面大量广的器件。在应用方面，它可以适应各种用途；在用量上，由于它用于很多廉价通用的产品，故用量极大。廉价单片机最大的应用场合一般是在家用电器和玩具等领域；另外，在交通、控制、仪器上也会发现它们的踪迹。

廉价单片机之所以便宜，首先在于它的制造过程简单及所用材料较省。这就意味着它的结构不会太复杂。显然，对于初步涉足单片机领域的人来说，它是一种利于学习和使用的器件。其次，由于它的产量较大，在工业化生产中就有较低的制造成本，故而其价格不高。低价格就容易被选中用于产品控制。再者，廉价也源于其工艺的成熟。生产单片机的集成电路工艺成熟，成品率就高，从而其价格较低。工艺的成熟，也使单片机有高度的可靠性，这十分有利于大批量、产业化产品的开发。

过去，人们虽然对单片机也给予重视，但对廉价单片机并没有一种明确而恰当的认识。为了使人们在工作中提高开发设计的水平，使所开发的产品更加经济、可靠，我们专门写了这本介绍廉价单片机的书。我们所遵从的宗旨是：在达到同一技术指标的条件下，最简单的结构是最科学的结构。

本书分为六章。第一章介绍廉价单片机的概况；第二章介绍 68HC05K 系列单片机；第三章介绍 68HC05J 系列单片机；第四章介绍廉价单片机的开发系统；第五、六章分别介绍 J，K 系列单片机的应用。

书中给出了一些十分实际、有效的应用例子。同时，为了使读者能用好廉价单片机，我们还专门介绍了一些特殊的技术方法。例如，采用 *RC* 电路配合软件完成 A/D 转换的技术，这些技术可以弥补廉价单片机内部没有 A/D 转换器的不足，同时又解决了实际中需要 A/D 转换的问题。

本书由余永权统筹，并且编写了第一、二章和第六章，林伟编写了第三、四章和第五章。

笔者在编写本书的过程中，还要兼顾教学和科研工作，所以，在繁忙之中不可避免会存在各种错误和不足，诚切希望广大读者批评指正。

作者　于广东工业大学

1997 年 12 月 8 日

目　　录

第一章　廉价型单片机及其用途 …………………………………………………… (1)

1.1　廉价型单片机的发展 …………………………………………………… (1)

1.1.1　单片机选择的原则 …………………………………………………… (1)

1.1.2　廉价型单片机的发展过程 ………………………………………… (3)

1.2　廉价型单片机的用途 …………………………………………………… (5)

第二章　MC68HC05K 系列单片机原理 ………………………………………… (8)

2.1　68HC05K 系列单片机结构概况 ……………………………………… (8)

2.1.1　68HC05K0/K1 主要性能 ………………………………………… (8)

2.1.2　MCU 的结构框图及引脚分布 …………………………………… (9)

2.1.3　外部振荡器连接的方法 …………………………………………… (10)

2.2　68HC05K 系列单片机的 CPU ………………………………………… (13)

2.2.1　CPU 中的寄存器 …………………………………………………… (13)

2.2.2　CPU 的标志及其意义 ……………………………………………… (15)

2.2.3　算术逻辑部件 ALU ………………………………………………… (16)

2.3　68HC05K 系列单片机的存储器 ……………………………………… (17)

2.3.1　存储器组织及地址分配 …………………………………………… (17)

2.3.2　ROM 区段及其作用 ……………………………………………… (17)

2.3.3　RAM 区段及其作用 ……………………………………………… (23)

2.3.4　I/O 寄存器区段 …………………………………………………… (23)

2.4　68HC05K 系列单片机的 I/O 端口 …………………………………… (24)

2.4.1　并行 I/O 端口 ……………………………………………………… (25)

2.4.2　多功能定时器 ……………………………………………………… (30)

2.5　中断、复位及节电方式 ………………………………………………… (34)

2.5.1　中断及中断系统 …………………………………………………… (34)

2.5.2　复位及复位状态 …………………………………………………… (39)

2.5.3　节电方式 …………………………………………………………… (42)

2.6　指令系统 ………………………………………………………………… (45)

2.6.1　寻址方式 …………………………………………………………… (45)

2.6.2　指令类型 …………………………………………………………… (50)

2.6.3　指令系统及指令说明 ……………………………………………… (53)

第三章　MC68HC05J 系列单片机原理 ………………………………………… (64)

3.1　MC68HC05J 系列单片机的结构 ……………………………………… (64)

3.1.1 MC68HC05J 系列基本特性 …… (64)
3.1.2 MC68HC05J 系列基本结构 …… (64)
3.1.3 MC68HC05J 系列的并行 I/O 端口 …… (68)
3.1.4 MC68HC05J 系列的复位系统 …… (70)
3.1.5 MC68HC05J 系列的中断系统 …… (72)
3.1.6 MC68HC05J 系列的多功能定时器 …… (73)
3.1.7 MC68HC05J 系列低功耗工作状态 …… (75)
3.2 MC68HC05J1A 单片机特点 …… (78)
3.2.1 MC68HC05J1A 存储器结构 …… (78)
3.2.2 MC68HC05J1A 的并行 I/O 端口 …… (78)
3.2.3 MC68HC05J1A 的中断系统 …… (80)
3.2.4 MC68HC05J1A 的掩膜选择 …… (81)
3.2.5 MC68HC705J1A 特性 …… (82)
3.3 MC68HC05J1 单片机 …… (83)
3.3.1 MC68HC05J1 存储器结构 …… (84)
3.3.2 MC68HC05J1 单片机的仿真 …… (85)
3.4 MC68HC05J2 单片机 …… (85)
3.4.1 MC68HC05J2 存储器结构 …… (85)
3.4.2 MC68HC705J2 单片机 …… (85)
3.5 MC68HC05J3 单片机 …… (87)
3.5.1 MC68HC05J3 存储器结构 …… (88)
3.5.2 MC68HC05J3 的 I/O 端口 …… (89)
3.5.3 MC68HC05J3 的 16 位定时器 …… (90)

第四章 廉价单片机的开发系统 …… (93)

4.1 MC68HC05EVS 开发系统 …… (93)
4.1.1 基板 PFB 的结构 …… (93)
4.1.2 MC68HC05J3 仿真模块 …… (94)
4.1.3 MC68HC05J 系列 HC05EVS 开发系统的软件 …… (96)
4.2 MMEVS05/08 开发系统 …… (102)
4.3 J1A 的编程器与在线软件仿真器 …… (109)
4.4 MC68HC705J2 编程器 …… (115)
4.5 K 系列在线软件仿真器 …… (117)

第五章 MC68HC05J 系列的应用 …… (123)

5.1 键盘矩阵接口设计 …… (123)
5.2 J 系列单片机与 RS232 通信 …… (130)
5.3 J 系列单片机与 EEPROM 接口 …… (139)

第六章　MC68HC05K 系列单片机的应用……………………………………………………（153）

6.1　68HC05K0 控制的蒸炖煲 ……………………………………………………………（153）

6.1.1　蒸炖煲的结构及蒸炖原理 ……………………………………………………（153）

6.1.2　蒸炖煲的工作要求及控制框图 ………………………………………………（154）

6.1.3　电路工作原理 …………………………………………………………………（156）

6.1.4　控制软件及操作状态 …………………………………………………………（158）

6.2　68HC05K0 构成的遥控器 ……………………………………………………………（160）

6.2.1　遥控器的基本原理 ……………………………………………………………（160）

6.2.2　遥控器的工作过程 ……………………………………………………………（163）

6.2.3　遥控器的控制软件清单 ………………………………………………………（165）

6.3　68HC05K0 控制的温度检测仪 ………………………………………………………（171）

6.3.1　温度检测仪的基本原理 ………………………………………………………（171）

6.3.2　外接 *RC* 实现 A/D 转换的方法 ………………………………………………（176）

6.3.3　温度检测仪电路及工作原理 …………………………………………………（190）

6.3.4　控制软件框图 …………………………………………………………………（195）

6.4　68HC05K0 控制的微工件抛光机 ……………………………………………………（199）

6.4.1　微工件抛光机的基本原理 ……………………………………………………（199）

6.4.2　直流电动机速度控制的方法 …………………………………………………（201）

6.4.3　控制电路及其工作原理 ………………………………………………………（209）

6.4.4　抛光机的控制软件框图 ………………………………………………………（214）

第一章　廉价型单片机及其用途

廉价单片机是指价格低廉的单片机。这种单片机一般结构较为简单，引脚数较少。所谓结构简单，不是指单片机内部的 CPU 简单，而是指单片机的内存容量较少，端口部件较少。MOTOROLA 公司的廉价单片机有 J，K 两个系列。这两个系列都是采用 68HC05 标准的 CPU，只是内存、定时器以及 I/O 端口等较少。

廉价单片机在各种功能简单的家用电器中，或者在功能单一、控制程序不长的仪器中应用十分广泛。

1.1　廉价型单片机的发展

廉价单片机、常规单片机以及高档单片机都是根据实际应用的需要推出的。高档单片机用于系统较复杂，功能较多，端口要求较多的场合，例如，变频控制系统，一台机床的控制系统。普通单片机用于有一定复杂程度和功能的系统，例如，洗衣机控制，加热炉温度控制等。廉价单片机用于被控对象功能较少，要求不多的场合，例如，水位控制，遥控器等。

人们常说："十个指头有长短。"这不仅用于说明人群中人的优劣，也用于说明不同的指头有不同的功用。毕竟，长短粗细是各有所功用的。同样，价格及性能不同的单片机亦有其不同的用途。

1.1.1　单片机选择的原则

针对一定的用途，恰当选择所用的单片机是十分重要的。中国有句古话为"杀鸡焉用牛刀"就是说做事要选用合适的工具。对于明确的对象，选择功能过少的单片机，无法完成控制任务；选择功能过强的单片机，则会造成资源浪费，使性能价格比下降。

在实际应用中，需要对单片机进行恰当的选择，选择的原则主要有如下几点。

一、对应用系统的适应性

所谓对应用系统的适应性，就是能否用一个单片机完成对系统的控制，或需要增加几个附加的集成电路才能实现对系统的控制。

从对应用系统的适应性出发，主要应考虑下列问题。

1. 单片机是否含有所需的 I/O 端口数目

如果单片机的 I/O 端口数太少，那么就不可能满足有关的功能。如果单片机的 I/O 端口数太多，就会造成单片机资源的浪费，也即选择了价格过高的单片机。

2. 单片机是否含有所需的外围端口部件

对于一个应用系统，首先要考虑的是，在 I/O 端口中有多少种不同类型的 I/O 方式。例如，是否包含下列 I/O 方式或 I/O 器件：

(1) RS232C 终端；

(2) 开关，继电器，键盘；

(3) 检测器，如检测温度、压力、光线、电压等；

(4) 声音报警器；

(5) 显示器，包括 LED，LCD 显示器；

(6) A/D 或 D/A 转换器；

(7) 其他器件或功能部件。

如果应用系统包含有上述器件或 I/O 方式，那么，必须考虑单片机内是否有能满足这些 I/O 方式或外部器件的电路部件。如果单片机不含有所需的电器部件或不满足所需的 I/O 方式，那么就无法满足系统需要。如果单片机含有比需要还多的部件或 I/O 方式，则该单片机超出了需要。

3. 单片机的 CPU 是否有合适的吞吐量

这是一个有关单片机计算功能的问题。针对应用系统的需要，必须认真考虑单片机对系统执行控制时的处理能力。如果单片机的处理能力过强，则浪费了单片机的资源；如果处理能力不足，就无法正常进行工作。

单片机的处理能力主要表现在 CPU 的运行速度，指令的功能，指令周期的长短，中断能力，堆栈大小等指标上。

4. 单片机的极限性能是否能满足要求

一个应用系统有其特定的应用环境，功耗和电压状态。单片机的极限性能一般包括最高使用温度、最低使用温度，最高使用电压、最低使用电压，最大功耗，最大电流等。例如，一个单片机给出的极限参数如下：

最大温度使用范围：－55 ℃～＋125 ℃；

最大电压使用范围：2.7 V～6 V；

最大电流：150 mA；

最大功耗：1 W。

很明显，必须考虑应用系统的使用温度是否在单片机最大温度范围之内，使用电压、电流和功耗是否在单片机极限指标之内。如果不是，那么单片机便不能满足应用系统的需要。这时，应该选择能满足应用系统需要的另一种型号的单片机。

二、单片机的可购买性

在单片机能够适合应用系统时，还应考虑这种型号的单片机的可购买性。可购买性包括下列几点。

1. 单片机是否可直接购买到

这是指单片机能否直接从厂家或其代理商处买到。购买的途径是否顺畅。

2. 单片机是否有足够的供应量

作为产品上使用的单片机，一般是用作产品的控制器，因此，单片机的需求量和产品数量是相一致的。这时，要求能购买到足够数量的单片机。这样，所选择的单片机才能满足产品的生产。

3. 单片机是否仍然在生产之中

选择单片机，应该选择那些仍然在生产之中的型号。已经停产的单片机是不能选用的，因为它已无后续供货能力，直接影响到产品的继续生产和生命力；同时，也会给人以一种过时的感觉，从而影响产品的新颖性。

4. 单片机是否在改进之中

这主要是看某种型号的单片机是否有新的版本推出或准备推出，这样才有利于产品的升级换代。也就是说，这种单片机仍然有着旺盛的生命力，并且在一定时期可以采用新版本对产品升级，从而达到事半功倍的效果。显然，对于准备推出新版本或有新版本的单片机，选择用于应用系统或产品具有较强的后劲。

三、单片机的可开发性

这是一个十分重要的因素。所选择的单片机是否有足够的开发手段，直接影响到单片机能否顺利开发，并且较快地应用于被控对象中。如果没有足够的开发手段，则相应的单片机型号是不宜选择用于有关应用系统的。对于被选择的单片机，应考虑下列开发工具。

1. 汇编程序

2. 编译程序

3. 调试工具

(1) 评价模块 EVM(Evaluation Module)；

(2) 在线仿真器；

(3) 逻辑分析工具；

(4) 调试监视程序；

(5) 源码级调试监视程序。

4. 在线示范板服务 BBS(Bulletin Board Service)

(1) 实时执行；

(2) 应用例子；

(3) 缺陷故障报告；

(4) 实用软件，包括自由汇编程序；

(5) 样本源码。

5. 应用支持

(1) 考虑是否存在专职的应用支持机构；

(2) 考虑是否存在应用工程师、应用技术人员或应用销售人员的支持；

(3) 应考虑支持人员的学识水平，以及支持人员是否真正对解决有关开发问题感兴趣；

(4) 支持人员和机构是否有便利的通信工具，能否及时得到支持。

四、单片机制造商历史

对于单片机制造商，主要应考虑如下几点。

(1) 胜任单片机设计的实证；

(2) 生产的优点和可靠性；

(3) 按时供货情况；

(4) 多年的营销情况；

(5) 财经报告。

根据上面四个原则对单片机进行选择，就可以选择出最能适用于应用系统的单片机，从而保证应用系统有最高的可靠性，最优的性能价格比，最长的使用寿命和最好的升级换代性。

1.1.2 廉价型单片机的发展过程

按照单片机选择的原则，对于一般较低档的产品或较简单的控制对象，应该选择廉价型的

单片机。

MOTOROLA 公司是美国著名的半导体生产厂家。1974 年,MOTOROLA 推出了其第一代微处理器 MC6800。以 MC6800 微处理为基础,该公司进行了单片机的研制,并在 1979 年开始生产单片机 MC6801。这种单片机和 Intel 公司的 8051 属于同一档次,主要是用来和 Intel 公司的 8051 系列进行市场竞争。

为了适应市场及应用的需要,MOTOROLA 公司在 1979 年又推出了功能比 MC6801 简单的 MC6805 单片机系列。

MC6801 单片机在 1983 年前后发展成为具有较高性能的 CMOS 单片机系列,即 MC68HC11 系列。同时,MC6805 单片机发展成为 MC68HC05 系列。

MC68HC05 系列的最早产品为 MC68HC05C4,在 1984 年前后又推出了 MC68HC05C8。80 年代末期,MOTOROLA 公司生产的单片机主要是 MC68HC05 系列中的 A,B,C,M 和 P 等有关型号。

80 年代末,针对市场上有关简单被控对象,MOTOROLA 公司推出了最早的廉价单片机 MC68HC05J1。这是该公司第一个引脚为 20 只的单片机。其程序存储容量为 1 KB,RAM 的容量为 64B,I/O 引脚数为 14,片内含一个 15 位的定时器。

在 MC68HC05J1 的基础上,MOTOROLA 公司后来又推出了 MC68HC05J2, MC68HC05J3 以及 MC68HC05J1A,从而形成了 J 系列的廉价单片机产品。J 系列单片机的有关型号和主要特性如表 1-1 中所示。

表 1-1 J 系列单片机

型　号	ROM 容量/B	RAM 容量/B	定时器/个	I/O 口/个	引脚/只
MC68HC05J1	1K	64	1	14	20
MC68HC05J1A	1K	64	1	14	20
MC68HC05J2	2K	112	1	14	20
MC68HC05J3	2K	128	1	14	20

J 系列单片机的主要特点有如下几点。

(1) 通用 68HC05 CPU;

(2) 内含 1 KB/2 KB 的用户 ROM;

(3) 内含 64 B/128 B 用户 RAM;

(4) 含有 14 个双向 I/O 端口;

(5) 有多功能定时器;

(6) 可以选用多种振荡器,如石英、陶瓷或 *RC* 振荡器;

(7) 有多种节电工作方式,如 STOP,WAIT 以及数据保存方式;

(8) 单一 3 V~5.5 V 电源;

(9) 在 2 V 电源的条件下可保存数据;

(10) 含有 COP Watchdog;

(11) 非法地址复位;

(12) 指令系统含有 8×8 无符号乘法指令;

(13) 采用 20 只引脚封装。

J 系列单片机可以满足对很多简单对象控制的需要，并且在实际应用中还有较多资源没有加以利用；就是说，对于某些简单对象控制只用到 J 系列单片机的一部分资源。这说明，还可以生产比 J 系列单片机更廉价、结构更简单的单片机。

K 系列单片机是 MOTOROLA 公司在 J 系列单片机推出之后，紧接着推向市场的更为廉价、更为简单的单片机。这种单片机只有 16 只引脚，I/O 端口只有 10 个，可以说是 MOTOROLA 公司生产的所有单片机中的小弟弟。K 系列单片机的用户 ROM 容量也很小，不超过 1KB。很明显，它只能用于那些功能较少、控制动作较少的对象。

K 系列单片机的有关型号如表 1-2 所示。

表 1-2　K 系列单片机

型　号	ROM 容量/B	RAM 容量/B	定时器/个	I/O 口/个	引脚/只
68HC05K0	0.5K	32	1,WDOG	10	16
68HC05K1	0.5K	32	1,WDOG	10	16
68HC705K1	0.5K	32	1,WDOG	10	16
68HC05K3	920	64	1,WDOG	10	16
68HC805K3	920	64	1,WDDG	10	16

K 系列单片机的主要特点为：

(1) 通用 68HC05 CPU；

(2) 含有 504 B/920 B 用户 ROM；

(3) 含有 32 B/64 B 用户 RAM；

(4) 有 10 个双向 I/O 端口，用软件可对端口进行下拉晶体管编程；

(5) 有 8 mA 灌入电流驱动能力；

(6) 含有 IRQ 外部中断，另外，PA0～PA3 可实现外部中断；

(7) 可采用石英、陶瓷、二端或三端电阻振荡器；

(8) 有 15 位的多功能定时器；

(9) 有多种节电方式，如 STOP，WAIT 和 HALT 方式；

(10) 指令系统有 8×8 无符号乘法指令；

(11) 采用 16 只引脚封装。

目前，MOTOROLA 公司生产的廉价单片机基本就是这 J，K 两个系列。这两个系列具有不同的特点，根据实际的需要，用户应该恰当地选择它们用于被控对象，以获得最优的性能价格比。

1.2　廉价型单片机的用途

廉价型单片机，顾名思义，就是价格低廉的产品。它可以用在价格低廉的设备、仪器、家电或便携式产品中充当控制器或心脏的角色。廉价的家电、便携式产品等产量一般比较大，这样，廉价单片机就有了广泛而大量的用场。

J，K 系列廉价单片机在各种廉价产品中有着不可忽视的应用潜力和前景。

廉价单片机 J,K 系列在应用中有明显的特点：

(1) 被控对象的主要功能较少；

(2) 廉价单片机控制时所显示的内容较少；

(3) 被控对象的结构体积较小；

(4) 被控对象的信息处理量较少；

(5) 被控对象的控制速度要求不高；

(6) 被控对象所需的输入信息不多；

(7) 被控对象有较大的产量。

以下是廉价单片机 J,K 系列的一些主要用途和应用领域。

一、玩具控制

J,K 系列单片机由于价廉，引脚数少，用于玩具控制是十分适宜的。一方面，玩具的价格不高，不可能使用昂贵的微控制器，而 J,K 系列单片机则可以在价格上满足玩具的经济性的要求。另一方面，玩具的动作有一定的限度，J,K 系列单片机可以十分方便、有效地控制玩具的各种动作。再有，现在的玩具正朝着智能化的方向发展，采用 J,K 系列单片机可以通过软件处理使玩具具有一定的智能。

J,K 系列控制的智能玩具可以有如下几类。

1. 智能洋娃娃

她可以有压痛叫喊，眨眼，发笑，自动行走等功能。

2. 智能航海模型

它具有自动碰物转弯，前行卡死则倒车，报警，直行加速，转弯减速等功能。

3. 智能动物

仿音鹦鹉，能跑动、吠叫的小狗，转头展翅的画眉等。

4. 智能汽车模型、火车模型

5. 智能玩具风扇

6. 智能玩具钟表

7. 智能儿童摩托车、吉普车

8. 智能计数工具

二、家用电器控制

有一些家用电器的功能较为简单，采用 J,K 系列单片机进行控制不但能满足需要，而且可以提高其智能水平。

J,K 系列单片机主要适用于控制下列家用电器。

1. 洗衣机控制

可用 J,K 系列单片机取代相关的机械式控制，使之进化为电脑控制的洗衣机。不过，这种洗衣机的功能和原机械型的功能相当。

2. 电冰箱控制

在电冰箱控制中，J,K 系列单片机主要用于温度控制，以取代老式的温度控制器。

3. 抽油烟机控制

4. 电熨斗控制

5. 抽湿机控制

6. 电风扇控制
7. 消毒柜控制
8. 电热水瓶控制
9. 遥控器

三、计算机外部设备控制

J,K 系列单片机在计算机系统的外围设备中用途也十分广泛。如：
1. 鼠标
2. 稳压电源控制
3. 磁盘机控制
4. 键盘控制
5. 打印机从控制器

四、工业控制

在工业系统中,J,K 系列单片机可以用于各种单一的控制环节或目标。这些环节有：
1. 物理量的检测及处理

如烟雾检测、温度检测、压力检测、湿度检测,以及对这些类型的物理量检测结果进行处理。

2. 位置控制
3. 可控硅等功率器件控制
4. 温度控制器
5. 塑料注塑机、吹塑机动作控制
6. 机床部件动作控制
7. 各种连续生产过程的环节控制

五、汽车控制

J,K 系列单片机可以在汽车控制中用于各种专门的控制。这些专用的控制有：
1. 汽车报警控制
2. 窗口动作控制
3. 安全气囊控制
4. 汽阀控制
5. 发动机点火控制
6. 制动系统有关部件控制
7. 冷气控制
8. 转向系统控制
9. 润滑系统控制

六、通信系统控制

在通信系统中,J 系列单片机可以用于各种部件或辅助系统中。如用于编码产生,错码鉴别,通信设备动作控制等。

七、其他领域各种简单控制

在其他领域,J,K 系列单片机可以控制各种简单任务。如诊病仪器,水位测控仪器,跳步机控制器等。

第二章　MC68HC05K 系列单片机原理

MC68HC05K 系列单片机是 MOTOROLA 公司生产的单片机家族中最低档次的单片机，因而在价格上也是极廉价的。由于 MOTOROLA 公司的 68HC05 系列单片机都包含 68HC05 通用 CPU，因此，内部的信息处理能力相近。不同系列的单片机，例如 68HC05J 系列、K 系列单片机，只是片内所集成的外围部件、存储器的容量大小和功能有所区别。这一章将介绍 MOTOROLA 公司的 68HC05K 系列单片机的原理。

2.1　68HC05K 系列单片机结构概况

MC68HC05K 系列单片机虽然是廉价的低档单片机，但它的结构和普通的单片机是一样的，即内部含有 CPU、存储器、I/O 端口等。MC68HC05K 系列单片机的存储器容量较小，I/O 端口功能较弱，而 CPU 的功能则和其他的 68HC05 单片机是一样的。68HC05K 系列单片机的主要型号是 68HC05K0 和 68HC05K1，可以说它们的性能在 68HC05 系列单片机中是最弱的，不过对于某些用途来说，它们的功能可以用作一个很好的控制器。这一节介绍它们的结构概况和主要性能。

2.1.1　68HC05K0/K1 主要性能

68HC05K0/K1 的主要性能可以用其 MCU 性能和掩膜任选项来表述。

一、68HC05K0/K1 的主要性能

(1) 通用 68HC05 CPU；

(2) 存储器映射 I/O 寄存器；

(3) 504 B 的用户 ROM，包括 8 个用户向量区；

(4) 32 B 用户 RAM；

(5) 64 位个性 EPROM/OTPROM(68HC05K1 专有)；

(6) 10 只双向 I/O 引脚，可用软件对下拉晶体管编程；

(7) 有 4 只 I/O 引脚能带 8 mA 灌入电流负载；

(8) 4 只 I/O 引脚有外部中断能力；

(9) 全静态操作，无最小时钟速度；

(10) 片内有振荡电路，可以外接石英晶体、陶瓷振荡器，用掩膜选择可以选二端或三端 *RC* 振荡器；

(11) 计算机正常工作监视器；

(12) 15 位多功能定时器，并含实时中断电路；

(13) 节电方式有停止、等待、暂停和数据保留方式；

(14) 有 8×8 位无符号乘法指令；

(15) 有多种封装形式，包括 16 脚 PDIP 和 16 脚 SOIC。

二、掩膜可选择的功能

(1) 开启或屏蔽计算机正常工作监视器；
(2) 外部中断采用边沿触发或者用边沿和电平触发；
(3) 开启或屏蔽 A 端口中的外部中断功能；
(4) 开启或屏蔽低电压复位功能；
(5) 开启或屏蔽 STOP 指令；
(6) 片内振荡电路由石英晶体、陶瓷振荡器或由 *RC* 电路激发；
(7) 二端 *RC* 或三端 *RC* 激发的振荡器；
(8) 开启或屏蔽端口 A，B 的可编程下拉电阻。

2.1.2 MCU 的结构框图及引脚分布

68HC05K0/K1 的结构框图如图 2-1 所示。它由 CPU、存储器、I/O 端口、定时器和 COP Watchdog、内部振荡器和低压复位电路等组成。

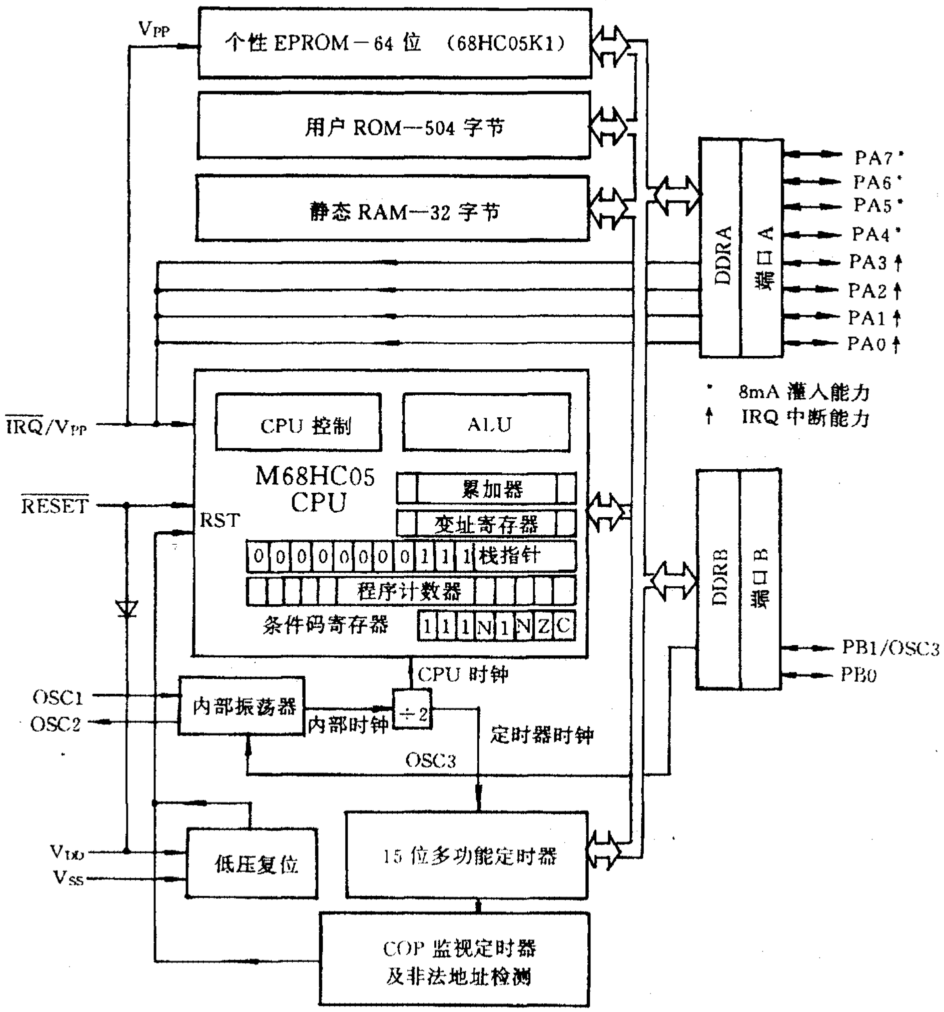


图 2-1 68HC05K0/K1 结构框图

68HC05K0/K1 的外形引脚分布如图 2-2 所示。图中，右边的是 PDIP 封装，左边的是 SOIC 封装。值得注意的是，它的电源引脚和一般通用的集成电路不同，是处于左边引脚的中部位置。

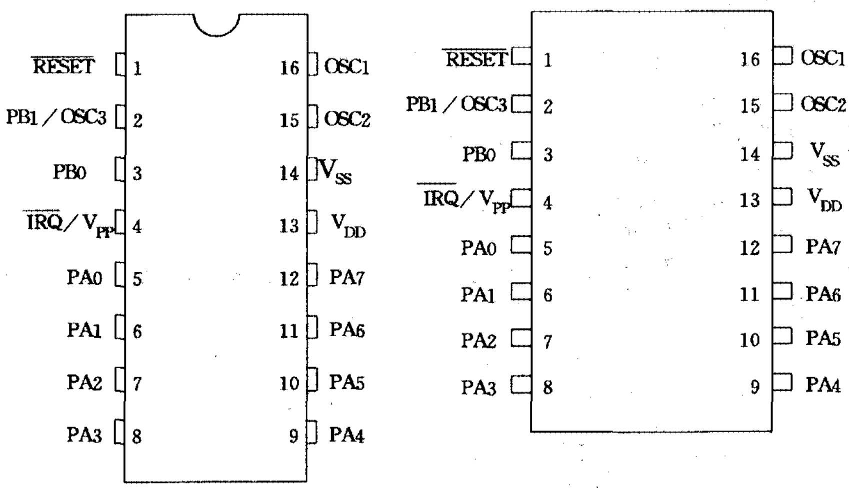


图2-2　外形封装和引脚分布

在图 2-2 中，68HC05K0/K1 的各条引脚的功能和作用如下：

1. 电源引脚

V_{DD}，13 脚：电源，+5V；

V_{SS}，14 脚：地。

2. 振荡器引脚

OSC1，16 脚：振荡器的引脚 1；

OSC2，15 脚：振荡器的引脚 2；

OSC3，2 脚：振荡器的引脚 3。

3. 复位引脚

$\overline{\text{RESET}}$，1 脚：外部复位输入引脚。低电平输入时会使单片机复位。

4. 中断引脚

$\overline{\text{IRQ}}$，4 脚：外部中断请求引脚。当请求信号为低电平时，单片机会响应中断。在掩膜选择边沿触发时，下降沿也会产生中断请求。

5. I/O 端口引脚

PA0～PA7，5～12 脚：这是 8 位并行 I/O 端口，可实现双向 I/O。

PB0～PB1，3，2 引脚：2 位双向并行 I/O 端口，其中 PB1 也用作振荡器引脚 3。

2.1.3　外部振荡器连接的方法

外部振荡器可连接石英晶体，陶瓷振荡器，二端 *RC* 振荡器，三端 *RC* 振荡器或接外部振荡信号。下面分别说明这些振荡器的连接方式。

1. 外部石英晶体振荡器

当把石英晶体连接到引脚 OSC1，OSC2 时，就可以驱动片内振荡器工作。当掩膜选择石英

晶体时，在片内的 OSC1 和 OSC2 两端连接了 2 MΩ 启动电阻。在石英晶体接到 OSC1，OSC2 端之后，石英晶体两端都应接 20pF～30pF 的小电容到地。而在硬件布线时，石英晶体和电容都应尽可能接近 OSC1，OSC2 引脚。连接情况如图 2-3 所示。

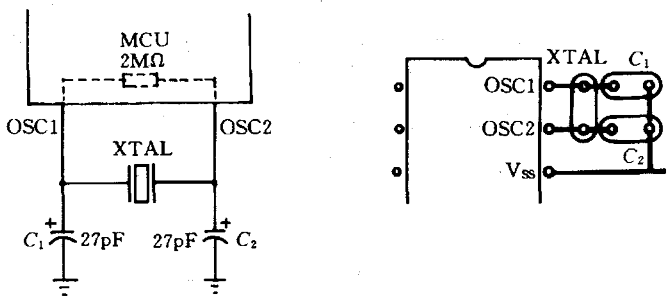


图 2-3　石英晶体振荡器

2. 外部陶瓷振荡器

陶瓷振荡器可以代替石英晶体作为外部振荡器。陶瓷振荡器有着比石英晶体更低的价格，故而经济性会更好。陶瓷振荡器的连接方式如图 2-4 所示。从图中可以看出，陶瓷振荡器不用连接电容，但是它的一端需要接地线。和石英晶体一样，陶瓷振荡器要尽可能和 OSC1，OSC2 连接，以保证振荡的稳定性和减少输出波形的畸变。

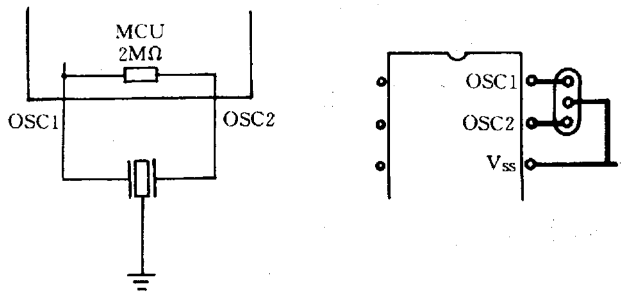


图 2-4　陶瓷振荡器

3. 二端 *RC* 振荡器

68HC05K0/K1 可以采用 *RC* 振荡器，它有着最低的价格，以获得最好的经济性。电阻、电容和 OSC1，OSC2 的连接情况如图 2-5 所示。电阻值、电容值和振荡频率的关系如下：

$$f_{OSC} = 1/(2.28RC)$$

其中，f_{OSC}是振荡器频率。

采用 *RC* 作为外部振荡元件，这时在 OSC1 处的信号波形是三角形，OSC2 处的信号波形是方波。对于 *RC* 振荡器来说，最佳振荡频率是 2 MHz。

于是，

$$f_{OSC} = 2\ \text{MHz} = 1/(2.28RC)$$

所以，

$$RC = 1/(4.56 \times 10^6)$$

当选定了电阻 *R* 之后，就可以求出电容值；反之，选定了电容 *C* 之后也可以求出电阻 *R*。

当 $R=10\ \text{k}\Omega$ 时，$C=22\ \text{pF}$；当 $C=51\ \text{pF}$ 时，则取 $R=4.3\ \text{k}\Omega$。

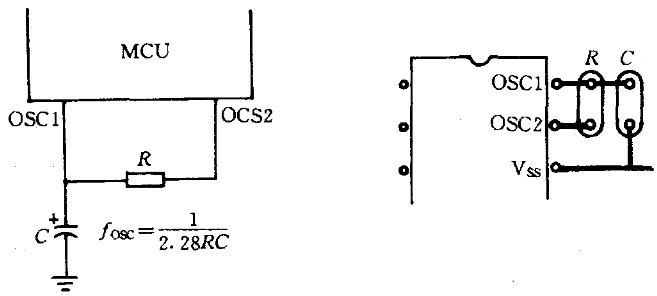


图 2-5　二端 *RC* 振荡器

4. 三端 *RC* 振荡器

三端 *RC* 振荡器的连接方式如图 2-6 所示。三端 *RC* 振荡器和二端 *RC* 振荡器一样，都是廉价的振荡器，但是三端 *RC* 振荡器无论在温度变化还是电源电压变化时都比二端 *RC* 振荡器稳定。

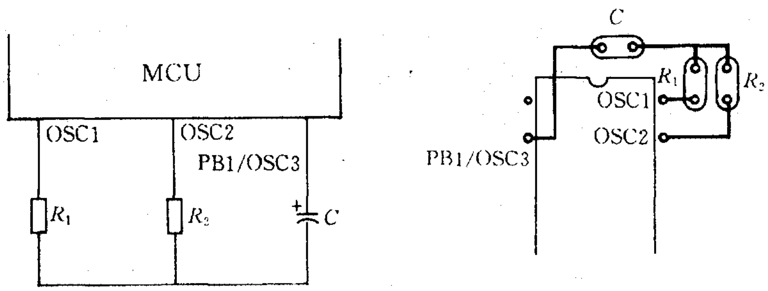


图 2-6　三端 *RC* 振荡器

三端 *RC* 振荡器的频率可以计算如下：

$$f_{osc}=1/(4.38RC)$$

在 OSC2 和 PB1/OSC3 引脚处的信号波形是方波，而在 OSC1 处的波形是三角波。在 OSC1 脚所连接的电阻 R_1 的阻值为 $0\ \Omega \sim 10\ \text{M}\Omega$。由于它在 *RC* 充放电周期中可减少由 OSC1 输入保护电路产生的断波效应，从而能改进精度。三端 *RC* 振荡器最佳的工作频率低于 1 MHz。在元件布置时，*RC* 元件应尽可能接近单片机的引脚 OSC1，OSC2 和 OSC3，以提高稳定性和减小输出畸变。

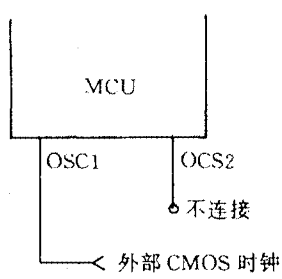


图 2-7　外部时钟的连接

5. 外部时钟信号作为振荡信号

在掩膜选择时，选择二端 *RC* 振荡器，则可允许外部 CMOS 兼容的时钟从 OSC1 输入并驱动内部振荡器。这时，OSC2 引脚可以留空或接一个 $10\ \Omega \sim 100\ \Omega$ 的电阻到地，这样可以减少无线电频率的干扰。外部时钟的连接如图 2-7 所示。

2.2　68HC05K 系列单片机的 CPU

68HC05K 系列单片机的 CPU 和其他任何一种计算机的 CPU 结构是一样的，包括控制器和运算器。控制器用于产生系统的所有控制时序及控制信号，包括指令的读/写、译码、产生所需的所有信号。运算器用于执行各种算术、逻辑操作。这一节将分别介绍 CPU 中的有关寄存器，CPU 工作的标志以及 CPU 中的运算部件(即算术逻辑部件)。

2.2.1　CPU 中的寄存器

在 68HC05K 系列单片机的 CPU 中有 5 个寄存器：累加器 A，变址寄存器 X，堆栈指针 SP，程序计数器 PC 和条件码寄存器 CCR。这 5 个寄存器和端口部件的寄存器不同，它们可以直接由指令执行操作或由控制器直接控制，故无需像端口部件的寄存器那样要由存储器映射方式读/写。

在 68HC05K 系列单片机 CPU 中，5 个寄存器的结构如图 2-8 所示。

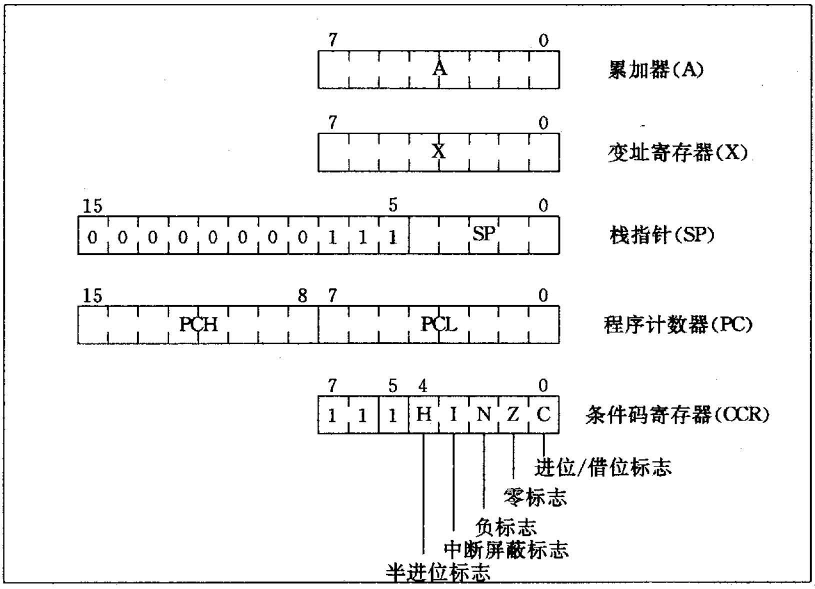


图2-8　CPU的寄存器

一、累加器 A

累加器 A 是一个通用的 8 位寄存器。在数据读取时，累加器用于存放从存储器读出的数据；在数据写入时，累加器用于存放准备写入存储器的数据。在执行算术、逻辑操作时，累加器首先存放一个操作数据；当操作执行完毕时则累加器存放操作结果。

当执行复位时，累加器的内容不受影响。

二、变址寄存器 X

变址寄存器 X 是一个 8 位寄存器。这个寄存器用于变址寻址方式中，用来确定操作数的地址。

在 68HC05K0/K1 中有三种变址寻址方式。这三种变址寻址方式分别是:无偏置量变址寻址,8 位偏置量变址寻址和 16 位偏置量变址寻址。

无偏置量变址寻址时,变址寄存器 X 存放操作数的低字节地址,CPU 自动用 $ 00 作为高字节地址。所以,无偏置量变址寻址方式可以对存储器地址 $ 0000～ $ 00FF 这 256 个单元进行寻址。这时,变址寄存器 X 中的内容即是存储器地址。因为 $ 0000～ $ 00FF 是 I/O 寄存器及 RAM 的地址,所以,在无偏置量变址寻址时,用变址寄存器 X 就可以对 I/O 寄存器或 RAM 进行访问。

8 位偏置量变址寻址时,指令中含有一个 8 位的偏置量。这时,操作数的地址是由变址寄存器 X 的内容加上偏置量之后形成的。所以,操作数地址范围为 $ 0000～ $ 01FE。

16 位偏置量变址寻址时,指令中含有一个 16 位的偏置量。这时,操作数的地址是由变址寄存器 X 的内容加上偏置量之和形成。所以,操作数地址范围为 $ 0000～ $ 03FF,即可以对存储器中的任何地址进行访问。

三、堆栈指针 SP

堆栈指针 SP 是一个 16 位的寄存器,它给出的是堆栈的下一个位置的地址。在执行堆栈复位指令 RSP 之后,堆栈指针的内容为 $ 00FF,这个地址即是栈底。

在从堆栈中弹出一个数据之前,堆栈指针 SP 的内容加 1;当把一个数据压入堆栈后,堆栈 SP 的内容减 1。很明显,弹出数据之后,堆栈指针指向栈底;压入数据之后,堆栈指针指向栈顶。

堆栈指针 SP 高 11 位的内容永远固定为“0000 0000 111”,如图 2-9 所示。

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit0
	0	0	0	0	0	0	0	0	1	1	1					
RESET:	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

图 2-9　堆栈指针 SP

堆栈指针 SP 的低 5 位内容是可以变化的,并根据数据出栈和入栈情况执行加 1 或减 1 操作。因此,堆栈指针所能表示的地址范围是从 $ 00E0 到 $ 00FF;也就是说,堆栈一共有 32 个位置。

调用一个子程序会占用 2 个堆栈位置,用于存放返回地址,即转子程序时程序计数器 PC 的内容。执行一次中断会占用堆栈的 5 个位置,用于存放条件码寄存器 CCR、累加器 A、变址寄存器 X 和程序计数器 PC 的内容。

子程序嵌套调用或中断嵌套响应有可能占用多于 32 个堆栈位置,这时,堆栈指针 SP 就循环计数到 $ 00FF 并从该地址开始进行压入数据的操作,这样就会把原来的数据覆盖掉,从而使这些返回数据及地址丢失,造成程序执行过程出错。为了保证程序的正确执行,不允许子程序嵌套调用和中断嵌套响应时压入堆栈的数据超过 32 个,也就是必须限制子程序嵌套调用或中断嵌套的深度。

复位时,堆栈指针 SP 的内容被置为 $ 00FF。

四、程序计数器 PC

程序计数器 PC 是一个 16 位的寄存器,用于存放下一条指令的地址、在执行转移指令时存放转移地址或者在中断时存放中断子程序入口地址。

68HC05K 系列单片机的内存容量小于 1 KB，其存储器的最大地址为 \$ 03FF。所以，程序计数器 PC 的高 6 位是无用的，被置为“000000”。程序计数器 PC 的结构如图 2-10 所示。

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit0
	0	0	0	0	0	0										
RESET：	0	0	0	0	0	0	装入地址 \$ 03FE 和 03FF 中的内容									

图 2-10 程序计数器 PC

每当执行一条指令时，程序计数器 PC 会自动加 1，以指向下一条指令的地址。在执行转子指令或中断响应时，程序计数器也会自动加 1，但加 1 之后的内容即下一条指令地址被压入堆栈，而程序计数器被装入子程序入口地址。在执行转移指令时，程序计数器被装入转移地址。

在 CPU 复位时，程序计数器被装入地址 \$ 03FE 和 \$ 03FF 中的内容。一般而言，\$ 03FE 和 \$ 03FF 的内容是复位入口地址，所以，在复位时，程序计数器中就存放了复位入口地址，以便从复位入口地址执行程序。

五、条件码寄存器 CCR

条件码寄存器 CCR 是一个 8 位寄存器，其作用是存放 CPU 执行有关操作时产生的标志信息。

条件码寄存器的结构如图 2-11 所示，D7，D6，D5 位均为 1；D4，D3，D2，D1 和 D0 位存放标志。高 3 位 D7～D5 永远固定为“111”；D4，D3，D2，D1 和 D0 分别存放半进位标志 H，中断屏蔽标志 I，负标志 N，零标志 Z 以及进位/借位标志 C。

	Bit7	6	5	4	3	2	1	Bit0
	1	1	1	H	I	N	Z	C
RESET：	1	1	1	U	I	U	U	U

U＝不受影响

图 2-11 条件码寄存器 CCR

CPU 复位时，条件码寄存器 CCR 中除 I 的内容外不受影响。也就说，条件码寄存器 CCR 在 CPU 复位时会把中断屏蔽标志 I 置“1”，而保留其他原来的标志内容不改变。

2.2.2 CPU 的标志及其意义

CPU 的标志存放在条件码寄存器 CCR 中。图 2-11 中给出了这些标志的符号及其存放情况。下面将对这些标志作详细说明。

一、半进位标志 H

半进位标志 H 在条件码寄存器 CCR 的 D4 位。当执行加法 ADD 或带进位加 ADC 操作时，如果参与运算的两个 8 位数据相加结果的低 4 位向上产生进位，则 CPU 把半位标志 H 置“1”。

半进位标志 H 对于一般运算是无用的，但在二进制编码的十进制(BCD)数据运算中十分有用。BCD 码运算是以四位二进制数来表示一位十进制数的，所以在 BCD 码算术运算中，半进位标志记录的是一位十进制数的进位。

二、中断屏蔽标志 I

把标志 I 置“1”,则可以屏蔽中断。

在中断屏蔽标志 I=0 时,如果有中断请求产生,则允许中断响应。这时,CPU 把累加器 A、变址寄存器 X、程序计数器 PC 和条件码寄存器 CCR 的内容压入堆栈,并把中断屏蔽标志 I 置“1”,随后把中断向量(即中断入口地址)取出来送入到程序计数器 PC 中。中断向量存放在 $03F8～$03FD 地址中。

当中断标志 I=1 时,如果有中断请求产生,则锁存中断请求。一般而言,一旦中断屏蔽标志 I 被清“0”,则 CPU 马上处理被锁存的中断请求。在中断服务子程序的尾部,常常会对中断屏蔽标志 I 清“0”。在允许中断嵌套的情况下,在进入中断服务子程序后,马上会对中断屏蔽标志 I 清“0”。

中断返回指令 RTI 会从堆栈中弹出 CPU 寄存器的内容,并且把中断屏蔽标志 I 清“0”。在执行 CPU 复位后,中断屏蔽标志被置为“1”,而它只能由软件指令清“0”。

三、负标志 N

在 CPU 执行算术运算、逻辑运算或处理数据时,如果产生负结果,则将负标志 N 置“1”。在数据运算和处理过程中,数据的最高位即 D7 用作符号位,如果 D7=1,说明数据为负;D7=0,说明数据为正。只要寄存器或存储器单元的 D7 位为 1,那么当把寄存器或存储器单元的内容送入累加器 A 时,就会使负标志 N=1。如果寄存器或存储器单元的 D7 位为 0,当把寄存器或存储器单元的内容送入累加器 A 时,就会使负标志 N=0。所以,负标志 N 可以用于检查有关寄存器或存储单元 D7 位的状态。

四、零标志 Z

当 CPU 执行算术操作、逻辑操作或数据处理的结果为 0 时,则将零标志 Z 置“1”;否则,对零标志 Z 清“0”。

五、进位/借位标志 C

当执行加法运算时,累加器的最高位 D7 产生进位,或者在执行减法运算时,需要 D7 向更高位借位,则 CPU 将进位/借位标志置“1”;否则清“0”。

某些逻辑操作和数据操作也会使进位/借位标志 C 置“1”或清“0”。

2.2.3 算术逻辑部件 ALU

算术逻辑部件 ALU 是 CPU 的核心部件。CPU 所执行的大多数操作都是在 ALU 中完成的,包括由指令系统所定义的算术操作和逻辑操作。

当 CPU 取出一条指令之后,就会对指令的操作码进行译码,译码之后就令 ALU 执行所选择的操作。大多数二进制算术运算是基于加法算法的,执行减法就是执行负数的加。乘法运算是在 ALU 内通过执行一连串的加法和移位操作完成的。乘法指令 MUL 需要 11 个内部处理器周期来完成这一连串的操作。

ALU 在执行加法等算术操作或逻辑操作时,是和累加器结合在一起协同完成的。例如加法运算,先将一个加数存放到累加器 A 中,而另一个加数在存储器单元中,把这两个数据从 ALU 的两个加法输入端送入 ALU 中,在完成相加并得出“和”之后,把“和”从 ALU 的输出端送回累加器 A 中。所以,累加器 A 总是和算术逻辑部件 ALU 联系在一起的。

2.3 68HC05K 系列单片机的存储器

68HC05K 系列单片机的寻址能力只有 1 KB，所以，它的存储器地址范围为 \$0000～\$03FF。在存储器地址范围之内分成多个不同的区段，每个区段的作用不同。这一节将介绍 68HC05K 系列单片机存储器的分配、组织以及各个区段的作用。

2.3.1 存储器组织及地址分配

68HC05K 系列单片机的 CPU 可以对 1 KB 存储器空间进行寻址。在读取一条指令或数据时，程序计数器 PC 自动加 1，从而指向下一条指令或数据。

存储器的组织及地址分配如图 2-12 所示。从图中可以看出，它包括 I/O 寄存器、堆栈、RAM、用户 ROM、测试 ROM 与 COP 寄存器、用户向量等区段。在 \$0000～\$03FF 地址中，还有一些区段是未用的，这些区段留给新的产品作有关存储区段。

存储器中最重要的是 ROM 区段。用户 ROM 用于存放程序指令和某些固定不变的数据，包括中断服务子程序和各种子程序。用户向量区段也是 ROM，用于存放用户的固定向量，例如，用户所定义的外部中断入口地址等。

I/O 寄存器是由存储器映射寻址的。这种寻址方式有很大的优点，使 CPU 对寄存器进行的读/写操作与对其他存储器单元的读/写过程一样，从而使操作的动作简化以及规则化，有利于系统设计及硬件集成的处理过程。

2.3.2 ROM 区段及其作用

在 68HC05K0/K1 单片机中，ROM 区段分成三个区间，具有不同的作用：用户 ROM、测试 ROM 和用户向量，对 68HC05K1 来说还有一个个性 EPROM/OTPROM。这些区间分别用于不同的目的。

一、用户 ROM

用户 ROM 有 496 个字节，地址范围为 \$0200～\$03EF。

用户 ROM 用于存放用户程序，包括主控程序、各种子程序、中断服务子程序。另外，用户 ROM 也用于存放一些固定不变的数据。用户 ROM 的空间较小，所以，无法存放复杂程序，这就决定了 68HC05K0/K1 单片机只能用于一些控制任务较为简单的对象。由于用户 ROM 的存储空间小，也就要求用户在编制程序时要尽量优化，使程序在完成某种功能的同时尽可能简短。

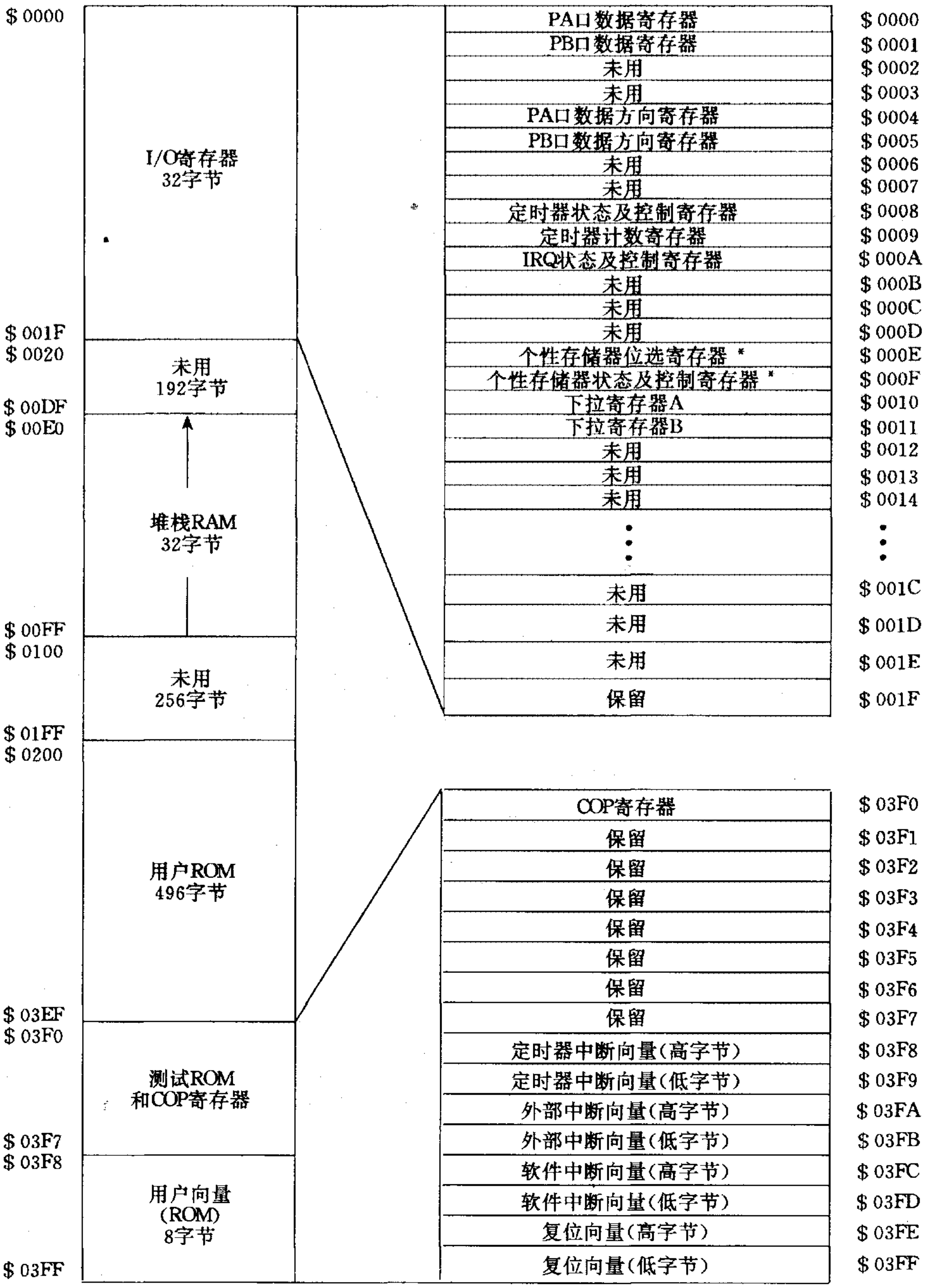
二、测试 ROM 和用户向量

这是两个区间，其作用不同。

1. 测试 ROM

测试 ROM 区间为 \$03F0～\$03F7。在这个区间中，\$03F0 用于计算机正常操作监视器 COP 寄存器和测试 ROM 数据。当把 0 写入到 \$03F0 的 D0(即 0 位)时，会对 MCU 中的 COP 寄存器清“0”。如果读 \$03F0 的内容，得到的是测试 ROM 数据。

在这个区间中，\$03F1～\$03F7 单元保留，待以后单片机改进时使用。



* MC68HC05K1独有

图2-12 存储器组织和地址分配

2. 用户向量

用户向量区间有 8 个单元,地址范围为 $ 03F8～ $ 03FF,共存放了 4 个用户向量:定时器中断向量、外部中断向量、软件中断向量和复位向量。这些向量的地址如下。

定时器中断向量,也是定时器中断入口地址,存放在地址 $ 03F8～ $ 03F9 中。其中,低位地址存放定时器中断向量的高位字节,高位地址存放低位字节。

外部中断向量，也就是外部中断入口地址，存放在地址＄03FA～＄03FB 中。其中，低位地址存放向量高位字节，高位地址存放低位字节。

软件中断向量，即软件中断入口地址，存放在地址＄03FC～＄03FD 中。

复位向量，即复位入口地址，存放在＄03FE～＄03FF 单元中。

三、个性 EPROM/OTPROM

个性 EPROM 简写为 PEPROM，是 68HC05K1 特有的存储器。它不包含在主存储器之内。

PEPROM 是由两个寄存器控制读/写的。这两个寄存器分别称为 PEPROM 位选择寄存器，PEPROM 状态和控制寄存器。这两个寄存器在存储器中的地址为＄000E 和＄000F。PEPROM 是一个特别的存储器，它的结构如图 2-13 所示。

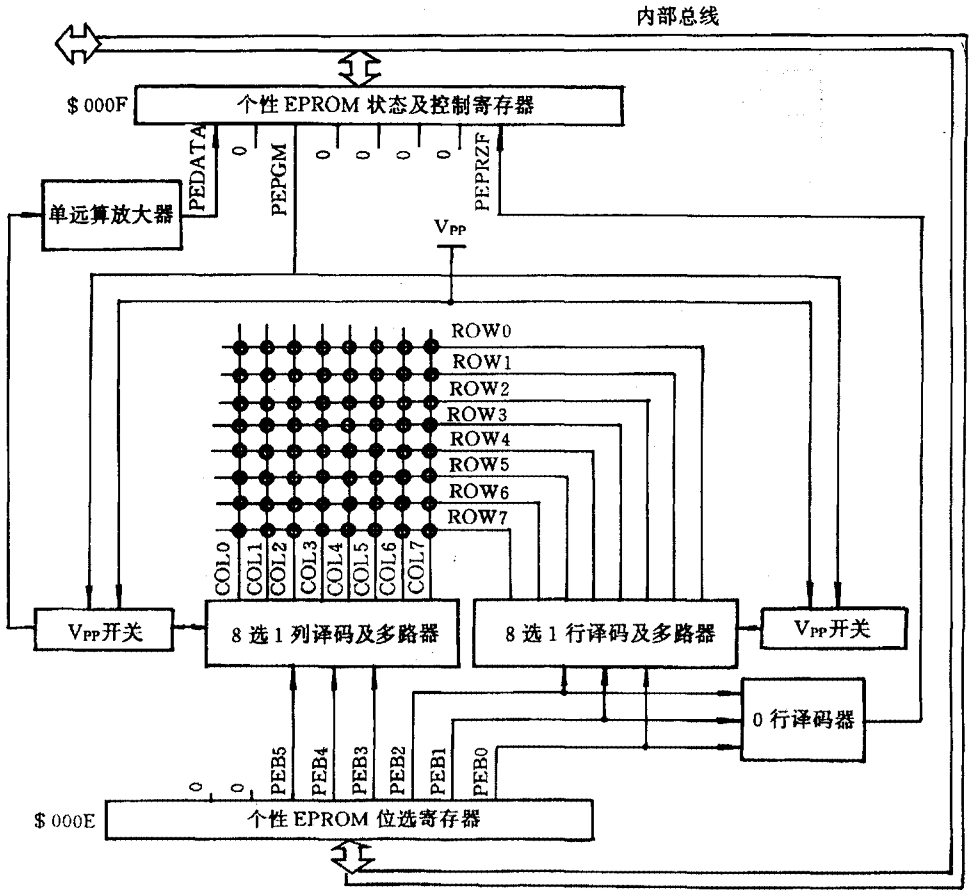


图 2-13　68HC05K1 的个性 EPROM

下面介绍一下 PEPROM 的工作原理。

1. PEPROM 位选择寄存器 PEBSR

PEPROM 位选择寄存器用于选择 PEPROM 阵列中 64 位中的 1 位。它是一个 8 位寄存器，在存储器中的地址为＄000E。其结构如图 2-14 所示。

	Bit7	6	5	4	3	2	1	Bit0
	PEB7	PEB6	PEB5	PEB4	PEB3	PEB2	PEB1	PEB0
RESET：	0	0	0	0	0	0	0	0

图 2-14 PEPROM 位选择寄存器

在 PEPROM 位选择寄存器中，PEB7，PEB6 未用，而其他位 PEB5～PEB0 用于选择 PEPROM 中的某一位。从图 2-13 可以看出，PEPROM 分成 8 行 8 列，即形成 64 位的矩阵。在 PEPROM 位选择寄存器中，PEB5～PEB3 用于选择列，PEB2～PEB0 用于选择行。位选择情况如表 2-1 中所示。

表 2-1 位选择情况

PEBSR	PEPROM Bit Selected
$ 00	Row0 Column0
$ 01	Row1 Column0
$ 02	Row2 Column0
⋮	⋮
$ 08	Row0 Column1
$ 09	Row1 Column1
$ 0A	Row2 Column1
⋮	⋮
$ 10	Row0 Column2
$ 11	Row1 Column2
$ 12	Row2 Column2
⋮	⋮
$ 38	Row0 Column7
$ 39	Row1 Column7
$ 3A	Row2 Column7
$ 3B	Row3 Column7
$ 3C	Row4 Column7
$ 3D	Row5 Column7
$ 3E	Row6 Column7
$ 3F	Row7 Column7

PEPROM 位选择寄存器在 CPU 复位时清“0”。它的 PEB7,PEB6 位虽然在 PEPROM 读/写时无用,但可以用作 68HC05K1 的存储器地址 $ 000E 的存储单元。

2. PEPROM 状态和控制寄存器 PESCR

PESCR 用于控制 PEPROM 的编程电压,并把读出的 PEPROM 位送入到内部数据总线。PESCR 还含有 0 行标志。PESCR 在存储器中的地址为 $ 000F,它的结构如图 2-15 所示。

	Bit7	6	5	4	3	2	1	Bit0
	PEDATA	0	PEPGM	0	0	0	0	PEPRZF
RESET:	0	0	0	0	0	0	0	0

图 2-15　PESCR 结构

PESCR 是一个 8 位寄存器,它的 D7,D5 和 D0 位用于 PEPROM 的状态标志和编程控制,其余位为 0。

在 CPU 复位时,PESCR 被清“0”。

下面说明 PESCR 的作用。

D7 位,称为 PEDATA 位,是一个只读位,表示 PEPROM 现行选择位的状态。

D5 位,称为 PEPGM 位,是一个编程控制位,可以被读/写。当 PEPGM=1 时,把编程电压 V_{PP}加到 PEPROM 被选中的位;当 PEPGM=0 时,则不加编程电压。

D0 位,称为 PEPRZF 位,是一个只读位。当 PEBSR 选中 PEPROM 阵列的 0 行时,则将 PEPRZF 位置“1”;当 PEBSR 选中其他行时,则将 PEPRZF 位清“0”。通过监测 PEPRZF 位可以减少存取 PEPROM 的一个字节所需的编码。

3. PEPROM 的编程

对 PEPROM 编程需要专门的软件,这种软件由 MOTOROLA 提供。

图 2-16 给出了对 PEPROM 进行编程的电路结构,这个编程电路使用 MOTOROLA 提供的编程软件。

用户也可以用自行编制的软件对 PEPROM 进行编程。在进行编程时,应把编程电压 V_{PP} 加到$\overline{IRQ}/V_{PP}$引脚。对 PEPROM 每一位编程的步骤如下:

(1) 通过把位选择数据写入 PEBSR 中来选中 PEPROM 中的 1 位;

(2) 将 PESCR 的 PEPGM 位置“1”;

(3) 等待 3 ms;

(4) 将 PESCR 的 PEPGM 位清“0”。

值得注意的是,在 PEPGM 位被置“1”和 V_{PP}已加在$\overline{IRQ}/V_{PP}$引脚时,不能对位执行读取操作,否则会使被读取的位不能编程。

4. PEPROM 的读取

读取 PEPROM 的有关位的步骤如下:

(1) 把位选择数据写入 PEBSR,选中 1 位;

(2) 读取 PESCR 中的 PEDATA 位;

(3) 把 PEDATA 位的内容存入 RAM 或寄存器;

(4) 改变 PEBSR 的内容以选择其他位;

(5) 连续存储 PEDATA 位,直到所需的 PEPROM 数据已被存储;

（6）连续读取和存储 PEPROM 数据。

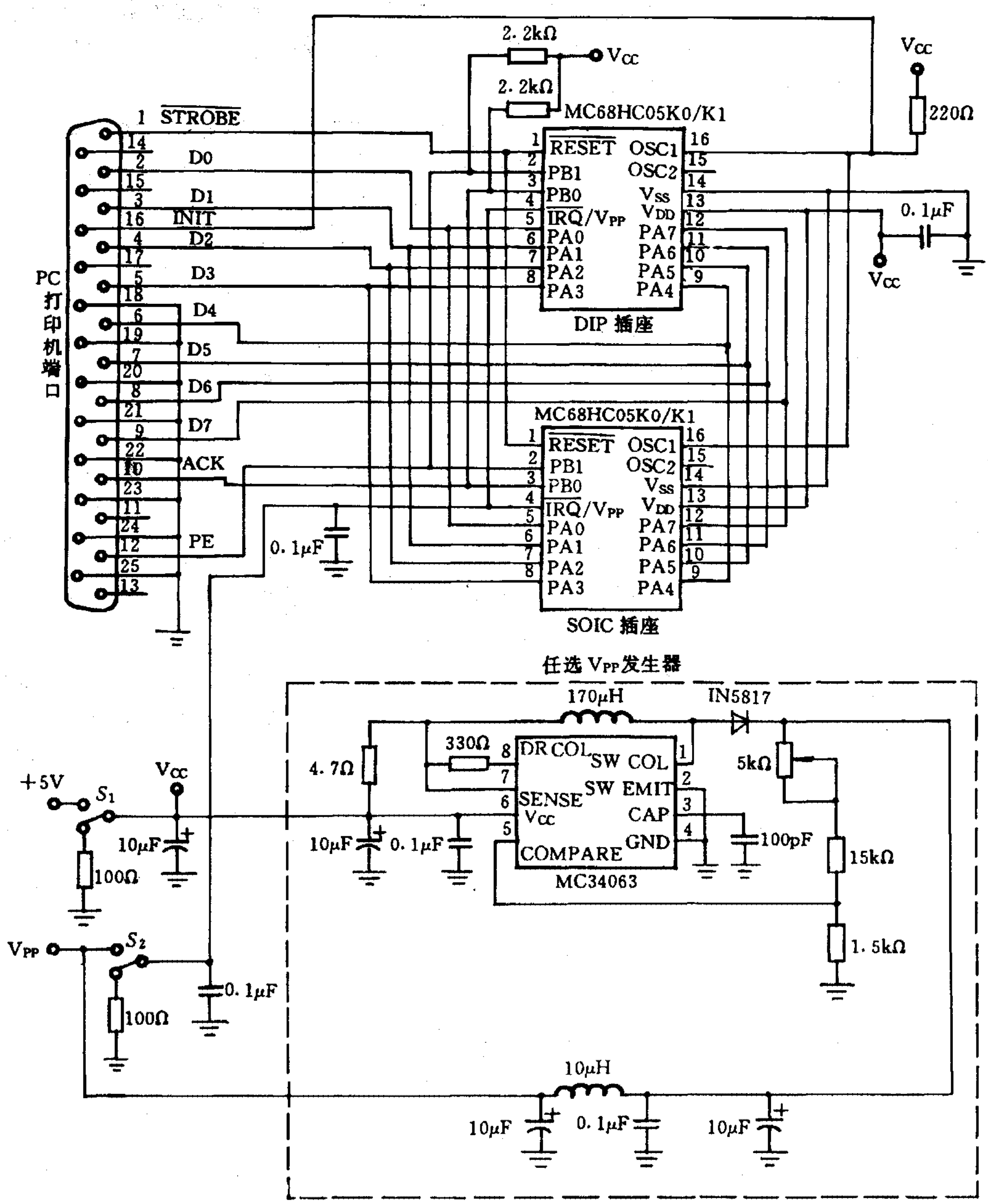


图 2-16　PEPROM 编程电路

当 PEPROM 的每一列含一个字节数据时，读取 PEPROM 是最容易的。这时，可以使列选择固定不变，从第 0 行开始选择位，每选中 1 位，就对 PEBSR 加 1，从而选择下一行的 1 位……一直下去，对 PEBSR 进行 7 次加 1 之后就会把同一列的一个字节读出。

当把一列的一个字节读出之后，可以改变列选择，通过把该列固定，从第 0 行到第 7 行进行位选择，又可以把这一列的一个字节读出。

由于当选择第 0 行时，PEPROM 状态和控制寄存器 PESCR 的 PEPRZF 位会被置“1”，因

此，只要监测 PEPRZF 位，就可以知道是否是一个字节的第一位。

5. PEPROM 的擦除

在封装中有窗口的 68HC05K1 允许用户用紫外线擦除 PEPROM。把 68HC05K1 暴露在波长为 2.537×10^{-7} m 的紫外线中便可擦除 PEPROM。一般把 68HC05K1 的窗口放在离紫外线约 2.54 cm 的地方较好。PEPROM 中被擦除的位的状态为 0。

没有窗口的 68HC05K1 中的 PEPROM 无法擦除，它只能用作一次性编程，即个性 OTPROM。

2.3.3 RAM 区段及其作用

在 68HC05K0/K1 中，RAM 位于地址 \$00E0～\$00FF 处。它既是用户 RAM，也是堆栈 RAM。所以，在 68HC05K0/K1 中，程序中的数据存储量不能太长。

当 RAM 用作堆栈时，在响应中断的过程中，CPU 会把寄存器内容压入堆栈，这时入栈的寄存器内容占用堆栈内 5 个字节位置。在调用子程序时，会把返回地址压入堆栈，这时会占用堆栈中 2 个字节的位置。执行堆栈操作时，堆栈指针会依据入栈或出栈自动进行减 1 或加 1 操作。

当把 RAM 用作堆栈时，栈底位于 \$00FF 处。这意味着，随着压入堆栈的数据的增加，数据将从栈底 \$00FF 往上朝着 \$00E0 处堆放。因此，对 RAM 进行一般数据存储就要小心，而且必须从 \$00E0 开始往高地址存放。

采用数据从 \$00E0 往 \$00FF 存储、堆栈从 \$00FF 往 \$00E0 堆放的方式，可以尽量使这两种数据隔开，从而避免相互覆盖。在 68HC05K0/K1 中，随机数据读/写和堆栈操作要特别注意，避免覆盖。

2.3.4 I/O 寄存器区段

在 68HC05K0/K1 的存储器中，存储空间的首部 32 个地址 \$0000～\$001F 是 I/O 区段，用于映射各种寄存器。这些寄存器包括端口数据寄存器、端口数据方向寄存器、与定时器有关的寄存器、与中断有关的寄存器、与 PEPROM 有关的寄存器、下拉寄存器等。

I/O 寄存器区段的寄存器及其对应地址、内容如图 2-17 所示。

I/O 寄存器区段的各种寄存器及其地址如下：

端口 A 数据寄存器：\$0000；

端口 B 数据寄存器：\$0001；

端口 A 数据方向寄存器：\$0004；

端口 B 数据方向寄存器：\$0005；

定时器状态及控制寄存器：\$0008；

定时器计数器寄存器：\$0009；

$\overline{IRQ}$状态及控制寄存器：\$000A；

PEPROM 位选择寄存器：\$000E；

PEPROM 状态和控制寄存器：\$000F；

下拉寄存器 A：\$0010；

下拉寄存器 B：\$0011；

其他地址中的单元不用或保留它用。

	Bit7	6	5	4	3	2	1	Bit0	
$ 0000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	端口A
$ 0001	0	0	0	0	0	0	PB1	PB0	端口B
$ 0002	—	—	—	—	—	—	—	—	未用
$ 0003	—	—	—	—	—	—	—	—	未用
$ 0004	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0	DDRA
$ 0005	0	0	0	0	0	0	DDRB1	DDRB0	DDRB
$ 0006	—	—	—	—	—	—	—	—	未用
$ 0007	—	—	—	—	—	—	—	—	未用
$ 0008	TOF	RTIF	TOIE	RTIE	TOFR	RTIFR	RT1	RT0	TSCR
$ 0009	Bit7	6	5	4	3	2	1	Bit0	TCNTR
$ 000A	IRQE	0	0	0	IRQF	0	IRQR	0	ISCR
$ 000B	—	—	—	—	—	—	—	—	未用
$ 000C	—	—	—	—	—	—	—	—	未用
$ 000D	—	—	—	—	—	—	—	—	未用
$ 000E	PEB7	PEB6	PEB5	PEB4	PEB3	PEB2	PEB1	PEB0	PEBSR
$ 000F	PEDATA	0	PEPGM	0	0	0	0	PEPRZF	PESCR
$ 0010	PDIA7	PDIA6	PDIA5	PDIA4	PDIA3	PDIA2	PDIA1	PDIA0	PDRA
$ 0011	—	—	—	—	—	—	PDIB1	PDIB0	PDRB
$ 0012	—	—	—	—	—	—	—	—	未用
⋮				⋮					
$ 001E	—	—	—	—	—	—	—	—	未用
$ 001F	—	—	—	—	—	—	—	—	保留
$ 03F0	—	—	—	—	—	—	—	COPC	COPR

图2-17 I/O寄存器

2.4 68HC05K 系列单片机的 I/O 端口

在 68HC05K 系列单片机中,I/O 端口有两个主要的部件,一个是并行 I/O 端口,一个是多功能定时器。这一节将分别介绍这两种端口部件。

2.4.1 并行 I/O 端口

68HC05K0/K1 的并行 I/O 端口有两个，一个是端口 A，另一个是端口 B。端口 A 有 8 只引脚，是 8 位并行双向 I/O 端口；端口 B 有 2 只引脚，是 2 位并行双向 I/O 端口。在这两个端口中，每一只 I/O 引脚都可以通过编程设置为输入或输出方式。编程是由数据方向寄存器决定，数据方向寄存器的内容确定每只 I/O 引脚的数据流动方向。

一、端口 A(PA)

PA 是一个 8 位通用双向 I/O 端口，它的特点如下：

(1) 在掩膜时任选可编程的下拉晶体管；

(2) PA7～PA4 这 4 只引脚有 8 mA 灌入电流能力；

(3) 在掩膜时，任选 PA3～PA0 这 4 只引脚作为外部中断输入。

PA 口的电路结构如图 2-18 所示。从图中可以看出，它由 PA 数据寄存器、数据方向寄存器 A、下拉寄存器 A、下拉晶体管以及有关逻辑门电路组成。

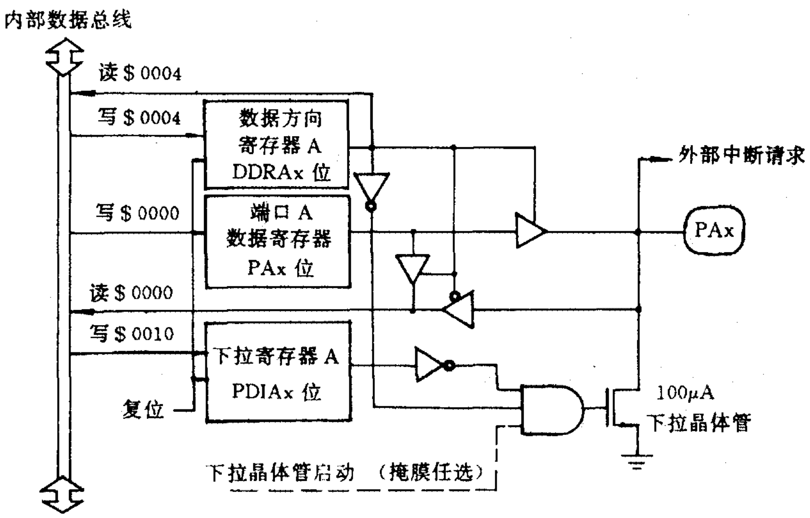


图 2-18 PA 口的电路结构

下面分别介绍 PA 口中的有关寄存器及其作用。

1. PA 数据寄存器

PA 数据寄存器是一个 8 位寄存器，其 8 位数据分别和 PA7～PA0 对应。它的最高位称为 PA7 位，次高位称为 PA6 位……最低位称为 PA0 位。

PA 数据寄存器的结构如图 2-19 所示。在 CPU 复位时，它的内容不受影响。PA 数据寄存器在存储器中的地址为 $ 0000。

	Bit7	6	5	4	3	2	1	Bit0
	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
RESET：	不受复位影响							

图 2-19 PA 数据寄存器

当 PA 口被编程设置为输出方式时，其数据寄存器的状态决定了输出引脚 PA7～PA0 的状态。当 PA 口被编程设置为输入方式时，如果读 PA 数据寄存器的内容，得到的是 PA7～PA0 引脚的状态。

CPU 通过 PA 数据寄存器将数据输出到引脚 PA7～PA0。但是，引脚 PA7～PA0 的输入数据是不通过 PA 数据寄存器而直接送入内部数据总线的。这一点从图 2-18 中可以清楚看到。

2. 数据方向寄存器 A(DDRA)

它用于确定 PA 引脚 PA7～PA0 为输入或输出方式。DDRA 的某位为“1”，则对应的 PA 口引脚为输出方式；DDRA 的某位为“0”，则对应的 PA 口引脚为输入方式。

数据方向寄存器 A 是一个 8 位寄存器，地址为 $0004，它的结构如图 2-20 所示。

	Bit7	6	5	4	3	2	1	Bit0
	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
RESET：	0	0	0	0	0	0	0	0

图 2-20 数据方向寄存器 DDRA

CPU 复位时，DDRA 的内容被清“0”，此时 PA 口的所有引脚为输入方式。

DDRA 的各位内容对 PA 口的控制作用如下：

DDRAi=1，则 PAi 为输出方式；

DDRAi=0，则 PAi 为输入方式。

当 PA 口被设置为输出口时，如果读 PA 口的内容，实际是在读 PA 数据寄存器的内容，而不是 PA 口引脚的电压。

3. 下拉寄存器 A(PDRA)

PA 口的所有引脚都具有掩膜任选可编程下拉晶体管，其电流为 100 μA。对下拉晶体管的编程是通过下拉寄存器 A 进行的。

下拉寄存器 A 的结构如图 2-21 所示。它是一个 8 位寄存器，地址为 $0010，它的每一位分别和 PA 口中相应引脚的晶体管对应。

	Bit7	6	5	4	3	2	1	Bit0
	PDIA7	PDIA6	PDIA5	PDIA4	PDIA3	PDIA2	PDIA1	PDIA0
RESET：	0	0	0	0	0	0	0	0

图 2-21 下拉寄存器 A(PDRA)

PDRA 中含有 PDIA7～PDIA0 这 8 个 PA 口下拉屏蔽位(Port A Pulldown Inhibit Bits)，它们是只写位。CPU 可以把数据写入 PDRA 中；如果读 PDRA，得到的是不确定数据。PDRA 中各位的意义如下：

PDIAi=1，PAi 引脚下拉晶体管开路。

PDIAi=0，PAi 引脚下拉晶体管接通。

注意不要对 PDRA 使用读指令，因为所得结果不对。

4. PA 口外部中断

如果在掩膜任选中选中了 PA 为外部中断，则 PA3～PA0 用作外部中断引脚。

外部中断的触发方式是掩膜选择的。PA3～PA0 可以是正边沿触发或者是正边沿和高电平触发。

应该注意，BIH 和 BIL 指令对外部中断测试时，是对$\overline{IRQ}/V_{PP}$引脚的电压进行测试，而不是对内部 IRQ 信号的状态进行测试。所以，BIH 和 BIL 指令不能对 PA 口的外部中断引脚进行测试。

PA 口引脚的功能如表 2-2 所示。

表 2-2 PA 口引脚功能

下拉掩膜选择	控制位		I/O 引脚方式	存取 PDRA		存取 DDRA	存取端口 A	
	PDRAx	DDIAx		读	写	读/写	读	写
No	X	0	输入高阻抗	U	PDIA7～0	DDRA7～0	引脚	PA7～0
No	X	1	输出	U	PDIA7～0	DDRA7～0	PA7～0	PA7～0
Yes	0	0	输入下拉	U	PDIA7～0	DDRA7～0	引脚	PA7～0
Yes	0	1	输出下拉	U	PDIA7～0	DDRA7～0	PA7～0	PA7～0
Yes	1	0	输入高阻抗	U	PDIA7～0	DDRA7～0	引脚	PA7～0
Yes	1	1	输出	U	PDIA7～0	DDRA7～0	PA7～0	PA7～0

说明：X＝不考虑；U＝未定义。

二、端口 B(PB)

端口 B(PB)是一个 2 位的双向 I/O 口。它和 PA 相似，可以通过数据方向寄存器编程设置为输出口或输入口。当被设置为输出口时，数据寄存器的内容被输出到 PB 引脚上；当被设置为输入口时，PB 引脚上的电平信号被输入到 CPU 内部。

在掩膜任选时选择三端 *RC* 振荡器，则屏蔽 PB1 下拉晶体管和数据方向寄存器的 DDRB1 位。PB1/OSC3 引脚用作振荡器的第三个控制端，同时，数据寄存器中的 PB1 位可用作一个存储位。

PB 口的特点如下：

(1) 掩膜任选时，可选择可编程下拉晶体管；

(2) 掩膜任选时，可选择三端振荡器连接方式。

PB 口的电路结构框图如图 2-22 所示。从图中看出，PB 口结构和 PA 口结构的每位是相同的，只是 PB1 在下拉晶体管的控制中多了一些条件，这些条件分别是三端振荡器和 *RC* 振荡器。

下面分别介绍 PB 口中有关寄存器及其工作原理。

1. PB 数据寄存器

PB 数据寄存器是一个 8 位寄存器，地址为 $0001。它只有最低 2 位有效，其高 6 位全为 0。在 PB 数据寄存器中，PB1 位和 PB1 引脚对应，PB0 位和 PB0 引脚对应。

PB 数据寄存器的结构如图 2-23 中所示。

PB 数据寄存器的 PB1 和 PB0 位是可以读/写的，而 PB7～PB3 不能写入，读出时全为 0。CPU 复位时并不影响 PB 数据寄存器的内容。

2. 数据方向寄存器 B(DDRB)

DDRB 用于控制 PB 口的方向，即为输入口或为输出口。DDRB 是一个 8 位寄存器，但只有低 2 位 DDRB1 和 DDRB0 有用，其余高 6 位无用。它的地址为 $0005。

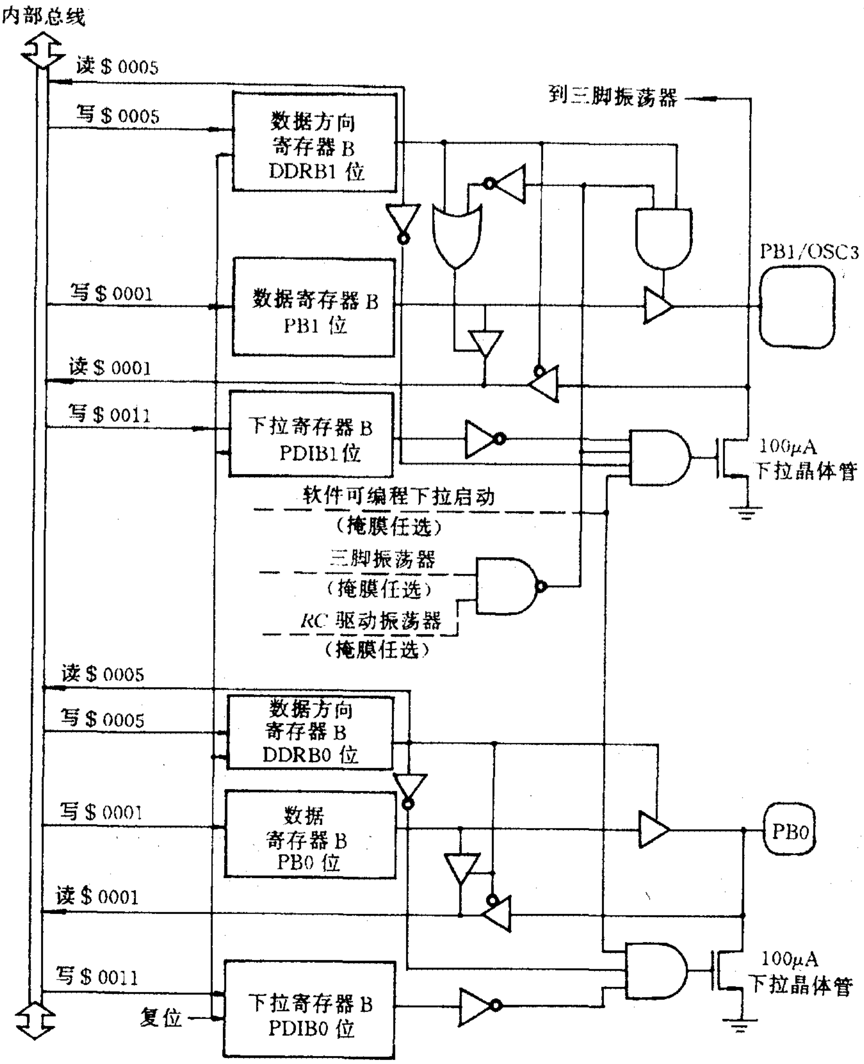


图 2-22　PB 口的电路结构

DDRB 的结构如图 2-24 中所示。

	Bit7	6	5	4	3	2	1	Bit0
	0	0	0	0	0	0	PB1	PB0
RESET：	不受复位影响							

图 2-23　PB 数据寄存器

	Bit7	6	5	4	3	2	1	Bit0
	0	0	0	0	0	0	DDRB1	DDRB0
RESET：	—	—	—	—	—	—	0	0

图 2-24　DDRB 的结构

把 DDRB 的某有效位写入“1”，则其对应的 PB 口引脚为输出脚；把“0”写入 DDRB 的某有效位，则对应引脚为输入脚。CPU 复位时，将 DDRB 清“0”，故 PB 口的 2 只引脚全为输入口。

DDRB 中的有效位作用如下：

DDRBi=1，则 PBi 为输出引脚，i=1，2。

DDRBi=0，则 PBi 为输入引脚，i=1，2。

在 DDRB 中，DDRB1，DDRB0 是可读/写的；但 DDRB7～DDRB3 是不能写入的，读出时结果为 0。

3. 下拉寄存器 B(PDRB)

PB 口的 2 位有掩膜任选可编程下拉晶体管，其典型电流为 100 μA。它们是通过下拉寄存器 PDRB 编程的。PDRB 的结构如图 2-25 所示。

	Bit7	6	5	4	3	2	1	Bit0
	—	—	—	—	—	—	PDIB1	PDIB0
RESET：	—	—	—	—	—	—	0	0

图 2-25　PDRB 的结构

PDRB 是一个 8 位寄存器，其中只有低 2 位 PDIB1，PDIB0 有效，其余 6 位无效。把 PDIB1，PDIB0 清"0"，则接通下拉晶体管。只有 PB 口是输入口时才能接通下拉晶体管，在为输出口时是不能接通下拉晶体管的。

CPU 复位时，对 PDIB1，PDIB0 清"0"。由于复位时也会把 PB 口编程为输入口，故这时会接通下拉晶体管。

PDRB 的地址为 $ 0011，它是只能写的寄存器，把"0"写入其中可接通下拉晶体管，把"1"写入其中可关断下拉晶体管。其意义如下：

PDIBi=0，PBi 接通下拉晶体管，i=1，2。

PDIBi=1，PBi 关断下拉晶体管，i=1，2。

由于 PDRB 是一个只写寄存器，故而对其读出时，其高 6 位得到的是 0，低 2 位得到的是不定数据。

如果 PB 口为输入口，读 PB 口时得到的是 PB 引脚的电平信号；如果 PB 口为输出口，读 PB 口时得到的是 PB 数据寄存器的内容。PB 数据寄存器是随时可以写入的，无需考虑数据方向寄存器 B 的状态。

表 2-3 和表 2-4 分别总结了 PB0 和 PB1/OSC3 两位端口的有关功能。

表 2-3　PB0 引脚功能

下拉掩膜选择	控制位		PB0 引脚方式	存取 PDRB		存取 DDRB	存取端口 B	
	PDIB0	DDIB0		读	写	读/写	读	写
No	X	0	输入高阻抗	U	PDIB0	DDRB0	引脚	PB0
No	X	1	输出	U	PDIB0	DDRB0	PB0	PB0
Yes	0	0	输入下拉	U	PDIB0	DDRB0	引脚	PB0
Yes	0	1	输出下拉	U	PDIB0	DDRB0	PB0	PB0
Yes	1	0	输入高阻抗	U	PDIB0	DDRB0	引脚	PB0
Yes	1	1	输出	U	PDIB0	DDRB0	PB0	PB0

说明：X=不考虑；U=未定义。

表 2-4 PB1/OSC3 引脚功能

掩膜选择		控制位		PB1/OSC3	存取 PDRB		存取 DDRB	存取端口 B	
三引脚振荡	下拉电阻	PDIB1	DDIB1	引脚方式	读	写	读/写	读	写
No	No	X	0	输入高阻抗	U	PDIB1	DDRB1	引脚	PB1
No	No	X	1	输出	U	PDIB1	DDRB1	PB1	PB1
No	Yes	0	0	输入下拉	U	PDIB1	DDRB1	引脚	PB1
No	Yes	0	1	输出下拉	U	PDIB1	DDRB1	PB1	PB1
No	Yes	1	0	输入高阻抗	U	PDIB1	DDRB1	引脚	PB1
No	Yes	1	1	输出	U	PDIB1	DDRB1	PB1	PB1
Yes	X	X	X	*RC* 振荡连接	U	PDIB1	DDRB1	PB1	PB1

说明:X=不考虑;U=未定义。

2.4.2 多功能定时器

68HC05K0/K1 中有一个可用来进行定时信号处理的部件,这就是定时器。定时器能够产生周期为某一特定时间的信号,可以用于时间控制,也可以用于监视程序运行,还可以产生某种特定的时钟。

多功能定时器主要分成两个部分:定时器和计算机正常运行(COP)监视器。多功能定时器的结构框图如图 2-26 所示。

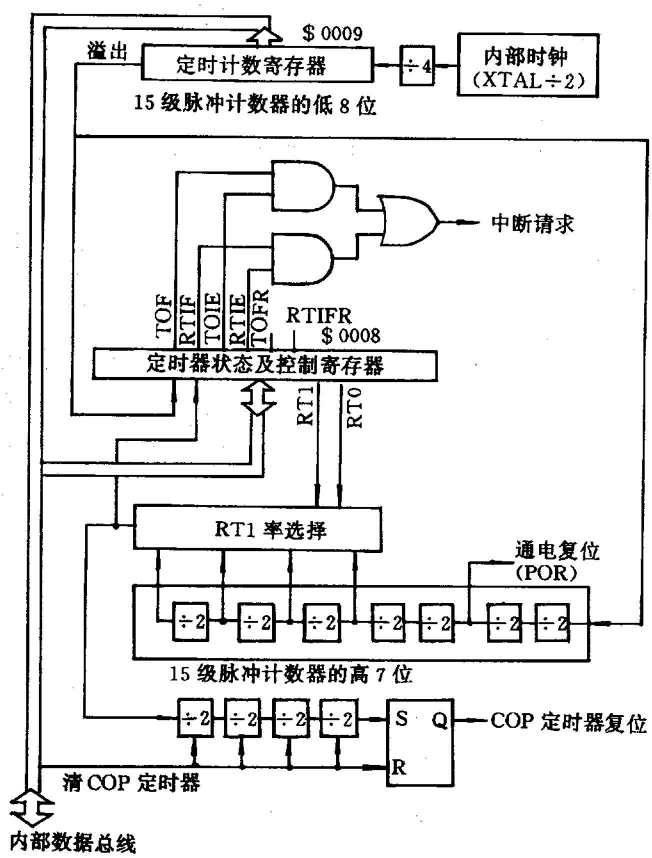


图 2-26 多功能定时器结构

一、定时器

定时器由定时器计数器寄存器、定时器状态控制寄存器、实时中断率选择、高 7 位脉冲计数器等组成。下面分别说明它们的作用。

1. 15 级脉冲计数器

15 级脉冲计数器是定时器的核心，它由低 8 位计数器和高 7 位脉冲计数器组成，在结构中它们是两个不同的部分。

15 级脉冲计数器的低 8 位计数器称为定时器计数器寄存器。它既是计数器又是一个 8 位寄存器，在存储器中的地址为 $ 0009。

定时器计数器寄存器是一个只读寄存器，其内容在任何时候是可读的。

定时器计数器寄存器简称 TCNTR，它的结构如图 2-27 所示。

	Bit7	6	5	4	3	2	1	Bit0
RESET：	0	0	0	0	0	0	0	0

图 2-27　定时器计数器寄存器

接通电源会使 15 级脉冲计数器清“0”，同时开始进行计数操作。振荡信号通过 2 分频之后产生内部时钟，内部时钟再经 4 分频之后进入定时器计数器寄存器 TCNTR 执行计数操作。在 4 064 个周期之后，上电复位电路释放，再次对 15 级计数器清“0”，并且使 MCU 脱离复位状态。

CPU 复位时，TCNTR 被清“0”。

因为内部时钟是经 4 分频之后进入 TCNTR 计数的，所以在 TCNTR 产生溢出信号(即已累计了 1 024 个内部时钟周期)时，溢出信号会产生定时器溢出(TOF)中断。故每经过 1 024 个内部时钟周期，会产生一次 TOF 中断。

高 7 位脉冲计数器是一个分级计数器，由 TCNTR 来的溢出信号 TOF 往上送到高 7 位脉冲计数器计数。高 7 位脉冲计数器中的第 2 位之后产生上电复位信号 POR，在第 4，5，6，7 位之后都有信号输出到实时中断率选择电路 RTIRS 中。

实时中断率选择电路是一个可以对高 7 位脉冲计数器的 4～7 位的信号进行选择的电路，选择方式由定时器状态控制寄存器 TSCR 的低两位 RT1，RT0 决定。选择不同位的输出信号，则中断周期不同。

2. 定时器状态控制寄存器 TSCR

TSCR 中含有以下状态及控制功能：

(1) 中断标志：定时器溢出标志，实时中断标志；

(2) 中断控制：定时器溢出中断开/关，实时中断开/关；

(3) 中断标志控制：定时器溢出标志复位，实时中断标志复位；

(4) 实时中断选择：实时中断选择 1 和 0 结合选择实时中断率。

TSCR 的结构如图 2-28 所示。

TSCR 在存储器中的地址为 $ 0008。CPU 复位时，TSCR 的高 4 位被清“0”，低 2 位被置“1”，其余 2 位不受影响。下面介绍 TSCR 每一位的意义。

Bit7	6	5	4	3	2	1	Bit0
TOF	RTIF	TOIE	RTIE	TOFR	RTIFR	RT1	RT0
RESET：0	0	0	0	U	U	1	1

U＝不受影响。

图 2-28　TSCR 的结构

TOF——定时器溢出标志

这是一个只读位。当定时器计数器寄存器 TCNTR 产生计数溢出，即计数从＄FF 变为＄00 时，置位 TOF 标志。

当允许定时器溢出中断位 TOIE 为 1 时，TOF 标志产生定时器溢出中断请求。要清除 TOF 标志，应对定时器溢出标志复位位写“1”，对 TOF 是不能执行写操作的。

RTIF——实时中断标志

这是一个只读位。当选择实时中断输出时，置位 RTIF 标志。

当允许实时中断位 RTIE 为 1 时，RTIF 产生实时中断请求。对实时中断标志复位位写“1”，则清除 RTIF 标志。RTIF 是不能写入的。

TOIE——允许定时器溢出中断位

这是一个可读/写的位，用于允许或屏蔽定时器溢出中断。

TOIE＝1，允许定时器溢出中断。

TOIE＝0，屏蔽定时器溢出中断。

RTIE——允许实时中断

这是一个可读/写的位，用于允许或屏蔽实时中断。

RTIE＝1，允许实时中断。

RTIE＝0，屏蔽实时中断。

TOFR——定时器溢出标志复位

这是一个只写位，对 TOFR 写“1”，则对 TOF 位清“0”。读 TOFR 时只会得到 0。

RTIFR——实时中断标志复位

这是一个只写位，对 RTIFR 写“1”则可对 RTIF 位清“0”。如果读 RTIFR，则得到的是 0。

RT1，RT0——实时中断选择 1 和 0

这是两个读/写位，可以组合起来对 4 种实时中断率进行选择。由于所选中的实时中断信号 RTI 用于驱动计算机正常运行监视器，即 COP Watchdog，所以，改变实时中断率同样改变 COP Watchdog 的计数率。

RT1，RT0 的组合情况及对应实时中断率选择如表 2-5 所示。表中同时给出了 COP 的监视周期时间。其中 f_{OP}为内部时钟频率。

二、COP 监视器

COP 监视器也称 COP Watchdog，是由 4 级计数器组成的，如图 2-26 所示。

COP 监视器是掩膜任选的。它用于检测软件的错误。在单片机中一般设定 COP 的运行周期，这个周期比单片机工作软件的运行周期稍长。在工作软件的尾部往往执行对 COP 的清“0”操作，这样，COP 永远无法计数到溢出。因此，若软件运行正常，则 COP 是不起作用的；但

当软件发生错误，其运行周期长于 COP 周期时，COP 就会发生溢出从而产生 COP 定时器复位信号，使 CPU 复位，于是又从软件首部开始执行。

表 2-5　实时中断率选择

RT1：RT0	RTI 率	RTI 周期/ms (f_{OP}=2 MHz)	COP 超时周期 (−0/+1 RTI 周期)	最小 COP 超时周期/ms (f_{OP}=2 MHz)
00	$f_{OP}\div 2^{14}$	8.2	8×RTI 周期	66
01	$f_{OP}\div 2^{15}$	16.4	8×RTI 周期	131
10	$f_{OP}\div 2^{16}$	32.4	8×RTI 周期	262
11	$f_{OP}\div 2^{17}$	65.5	8×RTI 周期	524

COP 寄存器用于控制 COP 监视器的工作。COP 寄存器简称 COPR，它的结构如图 2-29 所示。

	Bit7	6	5	4	3	2	1	Bit0
	—	—	—	—	—	—	—	COPC
RESET：	—	—	—	—	—	—	—	0

图 2-29　COPR 的结构

COPR 是一个 8 位寄存器，但只有最低位有用。它在内存中的地址是 $ 03F0。当 CPU 复位时，COPR 的最低位 COPC 被清“0”。

在 COPR 中，COPC 位是一个只写位，用于对 COP 监视器执行复位操作。如果读 $ 03F0，得到的是该地址中的 ROM 数据。

在运行、等待和暂停方式中，COP 监视器都起作用。STOP 指令会关断 COP 监视器。在通过 COP 监视器起作用的一些用途中，可以在掩膜任选时选择屏蔽 STOP 指令。

表 2-6 给出了允许或屏蔽 COP 监视器工作的条件。

表 2-6　COP 监视器工作条件

$\overline{IRQ}/V_{PP}$引脚电压	掩膜选择	等待/暂停时间	COP 工作情况
小于 $2\times V_{DD}$	STOP 指令屏蔽	小于 COP 超时周期	允许
小于 $2\times V_{DD}$	STOP 指令屏蔽	大于 COP 超时周期	屏蔽
小于 $2\times V_{DD}$	STOP 指令屏蔽	X	屏蔽
大于 $2\times V_{DD}$	X	X	屏蔽

说明：X=不考虑。

2.5 中断、复位及节电方式

中断系统是单片机实时工作的重要控制部件；复位是单片机初始化的功能和部件；节电方式是单片机满足各种特殊工作需要的工作方式。这一节将分别介绍中断系统、复位以及单片机的几种节电方式。

2.5.1 中断及中断系统

在程序正常执行的过程中，经常需要临时处理一些特殊的事件，这时就要调用中断程序而转入对特殊事件的处理。中断对于处理实时事件特别有用。在单片机中，中断是由中断系统执行的。

一、中断的类型

68HC05K0/K1 单片机有多种中断类型，分别是：

(1) 外部中断；

(2) 定时器中断；

(3) 软件中断。

无论哪一种中断都不会停止正在执行的指令操作，只有在当前指令执行完毕后中断才起作用。这样，既可以保证指令执行的完整性，又可以在下一条指令的开始转入中断处理。在中断响应时，首先把 CPU 寄存器的内容压入堆栈，并且在用户定义的中断向量地址中取出中断向量，然后将其装入程序计数器 PC，接着从中断向量指明的地址开始执行中断服务程序。

1. 外部中断

外部中断是 68HC05K0/K1 很重要的一种中断类型，它可以响应来自外部事件的请求。

外部中断有两种途径，一种是通过外部中断输入端 $\overline{IRQ}/V_{PP}$ 引脚进行外部中断请求。这时，加到 $\overline{IRQ}/V_{PP}$ 引脚的外部中断请求应该是逻辑信号“0”。另外一种途径是通过 PA3～PA0 输入外部中断请求。这时，要求输入的外部中断请求是逻辑信号“1”。通过 PA3～PA0 外部中断可以由掩膜任选进行选择。

2. 定时器中断

定时器中断有两种，一种是定时器溢出中断，一种是实时中断。

定时器溢出中断结构如图 2-26 中所示。定时器溢出中断是由 15 级脉冲计数器的低 8 位计数器，即定时器计数器寄存器 TCNTR 产生的。当 TCNTR 计数满并从 $FF 变成 $00 时，产生定时器溢出信号输出。定时器溢出中断周期只有一个，就是振荡器的 2 048 个振荡周期，即 1 024 个内部时钟周期。

实时中断的中断结构如图 2-26 所示。实时中断是由 15 级脉冲计数器产生的。通过定时器状态控制寄存器 TSCR 的低 2 位 RT1，RT0 以及实时中断率选择电路，实时中断可以选择 4 种不同的中断周期，分别为 2^{14}，2^{15}，2^{16} 或 2^{17} 个内部时钟周期。

3. 软件中断

软件中断是由指令 SWI 实现的。只要在程序执行中遇到软件中断指令 SWI，则 CPU 马上取其中断向量到程序计数器 PC，进入软件中断服务程序。

二、外部中断系统

外部中断系统用于对外部中断请求进行响应。外部中断系统的结构如图 2-30 所示。

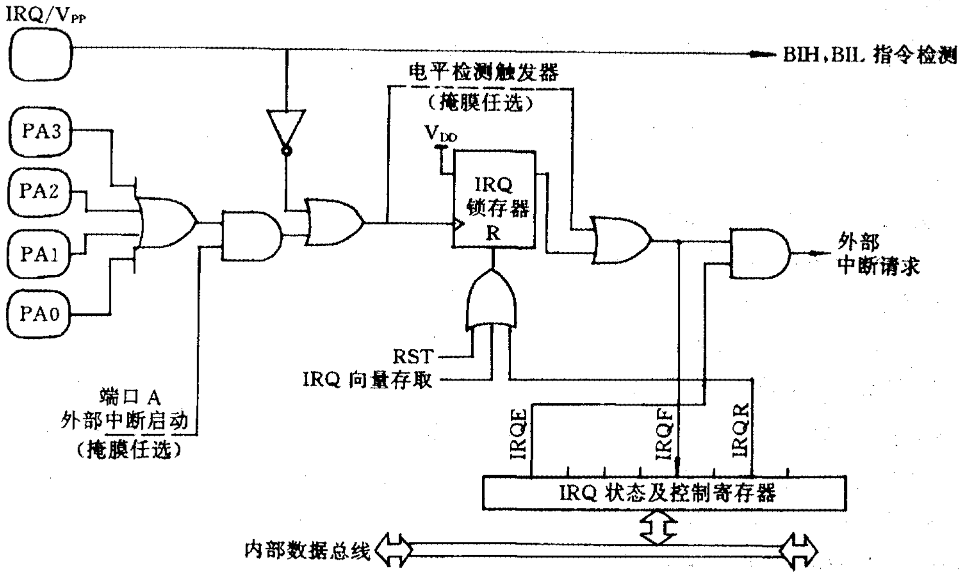


图 2-30 外部中断系统

从图 2-30 中可以看出，外部中断系统由中断状态及控制寄存器 ISCR，IRQ 锁存器以及有关逻辑门组成。

1. 中断状态及控制寄存器 ISCR

ISCR 是一个 8 位寄存器，在内存中的地址为 $ 000A，当 CPU 复位时，其最高位置“1”，其余位清“0”。

ISCR 中只有三位是有用的，它们分别是 D7——IRQE 位，D3——IRQF 位，D1——IRQR 位。ISCR 的结构如图 2-31 所示。

	Bit7	6	5	4	3	2	1	Bit0
	IRQE	0	0	0	IRQF	0	IRQR	0
RESET：	0	0	0	0	0	0	0	0

图 2-31 ISCR 的结构

下面分别介绍 ISCR 中有关位的作用。

IRQE——允许外部中断请求位

它是 ISCR 的 D7 位，该位可以进行读/写，用于允许或屏蔽外部中断。

IRQE＝1，允许外部中断。

IRQE＝0，屏蔽外部中断。

IRQF——外部中断请求标志位

ISCR 的 D3 位，该位在外部中断被悬挂起来时是只读位。IRQF 标志可以被清“0”。

下列两种情况之一将对 IRQF 标志置“1”：

(1) 在$\overline{IRQ}/V_{PP}$端出现逻辑“0”的外部中断请求信号时；

(2) 当 PA3～PA0 被掩膜选择为外部中断源时，在 PA3～PA0 任一端出现逻辑“1”的外部中断请求信号。

IRQF 的状态意义如下：

IRQF＝1，外部中断请求悬挂着，等待处理，即有外部中断请求。

IRQF＝0，无外部中断请求。

当 CPU 取出中断向量响应中断时，会清除 IRQF 标志。用户用指令写 IRQF 是无效的，若要清除 IRQF，可对 IRQR 位写“1”。

IRQR——中断请求复位位

是一个只写位，用于清除外部中断请求标志 IRQF。

IRQR＝1，清除 IRQF 位。

IRQR＝0，不影响 IRQF 位。

2. $\overline{IRQ}/V_{PP}$端的中断过程

$\overline{IRQ}/V_{PP}$输入的中断信号通过一个反相器和一个“或”门之后，被 IRQ 锁存器锁存。当 CPU 执行完当前指令后，对 IRQ 锁存器进行测试。如果 IRQ 锁存器的状态为“1”，说明有外部中断请求存在。随后 CPU 对条件码寄存器中的中断屏蔽位 I 以及中断状态及控制寄存器 ISCR 中的 IRQE 位进行测试，如果

I＝0，即允许中断，

IRQE＝1，即允许外部中断请求，

则 CPU 开始执行中断序列，即响应中断并转而执行中断服务子程序。

在响应中断时，CPU 首先取中断向量，这时，CPU 会把 IRQ 锁存器清“0”。因此，在执行中断服务子程序时可以锁存其他外部中断请求。

CPU 一旦响应中断，就会把中断屏蔽位 I 置“1”，封锁随后的中断请求。当 CPU 从中断返回时，中断屏蔽位 I 被清“0”，CPU 又可以响应新的中断请求。

外部中断的触发灵敏度是可以掩膜选择的。$\overline{IRQ}/V_{PP}$引脚可以是负边沿触发，也可以是负边沿及低电平触发。

当掩膜选择负边沿及低电平触发时，从$\overline{IRQ}$引脚输入的下降沿或低电平锁存外部中断请求。负边沿及低电平触发允许多个外部中断源以“线或”方式连接到$\overline{IRQ}/V_{PP}$引脚。一旦任何一个中断源令$\overline{IRQ}/V_{PP}$引脚为低电平，则外部中断请求被锁存。

当掩膜选择单纯的负边沿触发时，从$\overline{IRQ}$引脚输入的下降沿锁存外部中断请求。只有当$\overline{IRQ}/V_{PP}$引脚的电平返回“1”然后又下降为“0”时，随后的外部中断请求才可以被锁存。

$\overline{IRQ}/V_{PP}$端的中断逻辑如图 2-30 所示。

外部中断定时时序如图 2-32 所示。图中，上面的波形是单纯负边沿触发时的中断定时。这时，要求从$\overline{IRQ}$引脚输入的信号是负脉冲，脉冲宽度在电源为 5 V 时不得小于 125 ns，电源为 3 V时不得小于 250 ns；前后两个负边沿之间的时间不得小于执行中断服务子程序的时间与 19 个 t_{CYC}之和。其中，t_{CYC}是内部时钟周期。

在图 2-32 中，下面的波形是负边沿及低电平触发时的中断定时。其中，$\overline{IRQ_1}$，…，$\overline{IRQ_n}$ 是指在$\overline{IRQ}/V_{PP}$引脚用“线或”相连的多个中断源。如果$\overline{IRQ}/V_{PP}$引脚保持低电平，则在中断服务

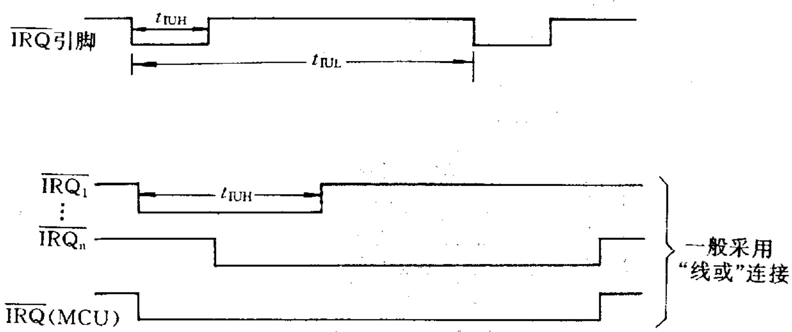


图 2-32 外部中断定时

子程序执行完毕之后，下一个中断又被识别。

3. PA3～PA0 端的中断过程

当掩膜选择 PA 口为外部中断输入口时，PA 口中的 PA3～PA0 用作附加的外部中断源。

从 PA3～PA0 引脚输入的中断信号锁存外部中断请求。这一点可以从图 2-30 中断逻辑中看出。

PA3～PA0 的中断过程和 $\overline{IRQ}/V_{PP}$ 端的中断过程一样。

三、定时器中断系统

定时器中断系统如图 2-26 所示。定时器可以执行下列中断：

(1) 定时器溢出中断；

(2) 实时中断。

对条件码寄存器的 I 位置“1”，则屏蔽定时器中断。

1. 定时器溢出中断

在定时器的 15 级计数器的低 8 位，也即定时器计数器寄存器 TCNTR 对内部时钟计数并产生溢出信号 TOF 时，如果定时器状态及控制寄存器 TSCR 中的允许定时器溢出中断位 TOIE＝1，则产生定时器中断请求。

2. 实时中断

当定时器中 15 级计数器的计数结果由实时中断率选择电路选择时，则置位实时中断请求标志 RTIF，即 RTIF＝1。如果 TSCR 中的允许实时中断位 RTIE＝1，则产生实时中断请求。

四、中断的处理过程

在响应中断之后，CPU 执行下列任务。

(1) 在堆栈中存放 CPU 寄存器的内容，存放顺序如图 2-33 所示，即从低地址到高地址按条件码寄存器 CCR、累加器 A、变址寄存器 X、程序计数器高字节 PCH、程序计数器低字节 PCL 的顺序存入；

(2) 把条件码寄存器中的中断屏蔽位 I 置“1”，禁止下一个中断；

(3) 把对应中断的向量地址的内容装入程序计数器 PC。不同的中断向量地址如下：

软件中断向量：$03FC～$03FD

外部中断向量：$03FA～$03FB

定时器中断向量：$03F8～$03F9

中断返回指令 RTI 会使 CPU 恢复存放在堆栈中的 CPU 寄存器的内容，即把堆栈中存放

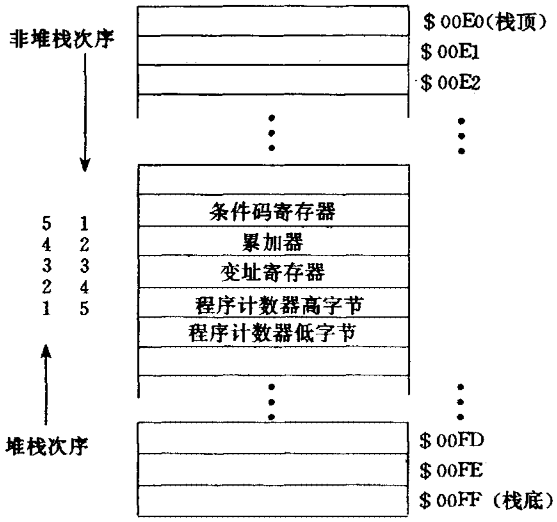


图2-33 中断时堆栈的顺序

的原 CPU 寄存器内容取出之后送回到 CPU 寄存器中。

中断的流程如图 2-34 所示。

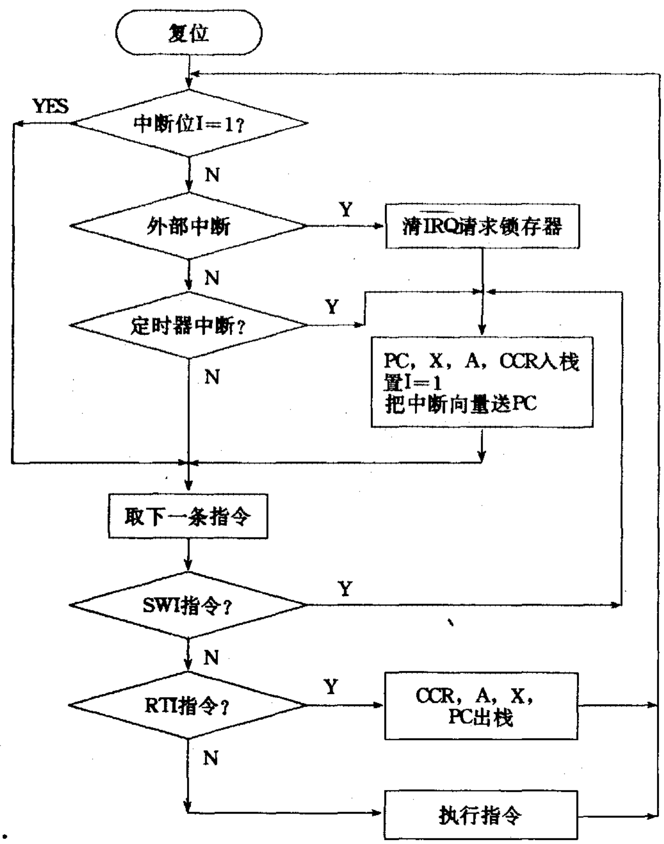


图2-34 中断的流程

表中 2-7 给出了复位、中断源以及向量的地址分配情况。在表中，COP Watchdog、低电压复位功能和 PA 口外部中断能力都是由掩膜任选选择的。

表 2-7　复位、中断源及向量分配地址

功　能	源部件	局部屏蔽	总体屏蔽	优先权 (1=最高)	向量地址
复位	通电 复位引脚 COP 低压 非法地址	None	None None None None None	1 1 1 1 1	\$03FE～\$03FF
软件中断 (SWI)	User Code	None	None	和指令优先权同	\$03FC～\$03FD
外部中断	$\overline{IRQ}/V_{PP}$ PA3 PA2 PA1 PA0	IRQE 位	1 位	2	\$03FA～\$03FB
定时器中断	TOF 位 RTIF 位	TOFE 位 RTIE 位	1 位	3	\$03F8～\$03F9

2.5.2　复位及复位状态

复位就是初始化操作。68HC05K0/K1 单片机有多种复位方法。由于 68HC05K0/K1 的内部结构有其本身的特点，故在执行复位之后其内部部件状态会处于特定的状态。这里将介绍 68HC05K0/K1 的复位方法以及复位后相关部件的状态。

一、复位的类型

复位会马上停止当前执行的指令操作，对一些控制位和寄存器内容初始化，同时把用户定义的复位向量装入程序计数器。复位之后，程序将从复位向量指明的地址开始执行，复位向量指明的地址也就是主程序入口地址。

68HC05K0/K1 的复位源一共有 5 个，如图 2-35 所示。这 5 个复位源分别是：

(1) 上电复位；

(2) 外部复位；

(3) COP 复位(掩膜任选时选择)；

(4) 非法地址复位；

(5) 低电压复位(掩膜任选时选择)。

下面分别对这 5 种不同的复位源进行介绍。

1. 上电复位

上电复位也称初始电源电压上升复位。它是在 68HC05K0/K1 的 V_{DD}引脚有正的电平变化时产生的复位。

上电复位是严格按电源电压上升所产生的复位，它不能用于检测电源电压的下降。

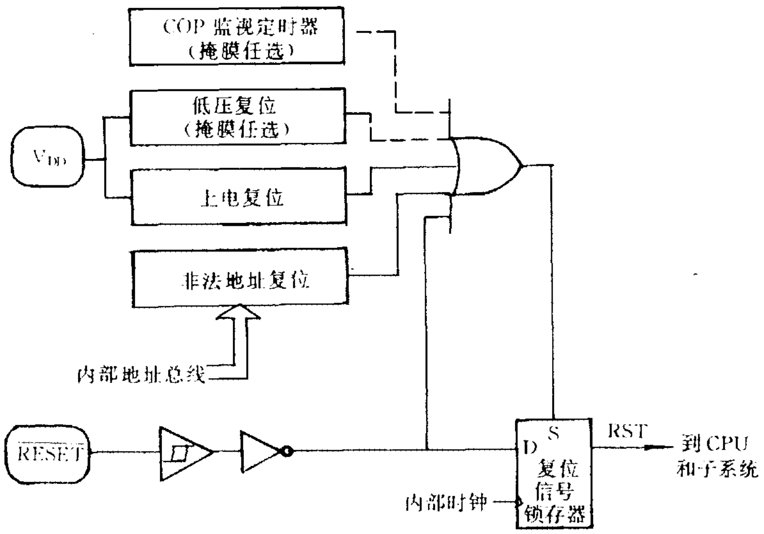


图 2-35 复位源框图

在振荡器启动之后经 4 064 个内部时钟周期 t_{CYC} 延时，时钟发生器到达稳定状态。如果在振荡器稳定之后，$\overline{RESET}$ 端处于逻辑“0”状态，那么，单片机会保持在复位状态直到 $\overline{RESET}$ 端信号上升为逻辑“1”。

上电复位的定时时序如图 2-36 所示。

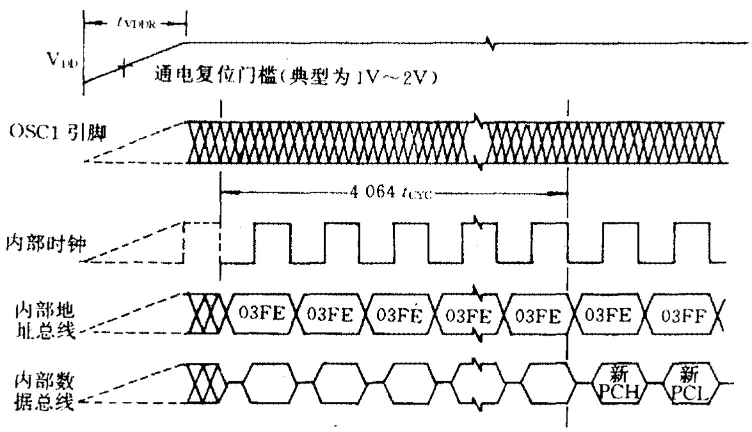


图 2-36 上电复位定时时序

2. 外部复位

外部复位实质上是在 $\overline{RESET}$ 引脚加上逻辑“0”信号所产生的复位。在外部复位时，要求加在复位引脚 $\overline{RESET}$ 上的逻辑“0”电平的保持时间大于 1.5 个 t_{CYC}。在电路结构中，用一个施密特触发器对 $\overline{RESET}$ 引脚的电平进行检测整形。

外部复位的定时时序如图 2-37 所示。

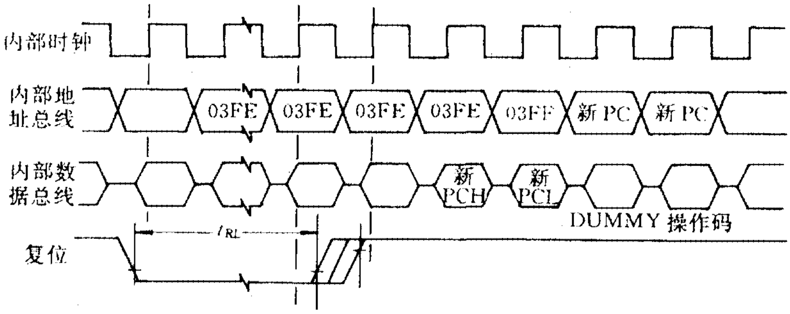


图 2-37　外部复位的定时时序

3. COP 复位

COP 复位是计算机正常操作复位。其本质是 COP Watchdog 定时器溢出产生的复位。

COP Watchdog 是检测软件错误的部件，它的定时器周期比程序周期稍长，在程序执行中如果对 COP Watchdog 进行正常清“0”，并使它重新计数，则 COP Watchdog 永远不会溢出，故而不会产生复位。如果程序不正常执行，使执行时间长于 COP Watchdog 计时周期，则 COP Watchdog 会溢出从而产生复位。

在程序正常执行时，为了清“0” COP Watchdog、防止 COP 复位，就要对地址 $ 03F0 的 COP 寄存器的 0 位(即 COPC 位)清“0”。

COP 寄存器的结构如图 2-38 所示。

	Bit7	6	5	4	3	2	1	Bit0
	—	—	—	—	—	—	—	COPC
RESET:	—	—	—	—	—	—	—	0

图 2-38　COP 寄存器 COPR

从图 2-38 中看出，COP 寄存器 COPR 是一个 8 位寄存器，但是只有最低位有用，其余高 7 位无用，COPR 的最低位 D0 称为 COPC 位。

COPC 位是一个只写位，称为 COP 清除位。把逻辑“0”写入 COPC 位，可对 COP Watchdog 清“0”。

如果对 COPR 执行读操作，读到的是地址 $ 03F0 中固化了的内容，因为 $ 03F0 是 ROM 中的一个地址。

4. 非法地址复位

如果从 ROM 和 RAM 以外的地址取操作码，那么称该地址为非法地址，这时会产生非法地址复位。

ROM 的地址为 $ 0200～ $ 03FF，RAM 的地址为 $ 00E0～ $ 00FF。取操作码必须在这两个地址区域之内，否则便是非法地址。

非法地址复位可以防止程序跳出正常存储器范围。

5. 低电压复位

低电压复位是掩膜任选时选择的。如果 V_{DD}引脚的电压下降到低于 3.5 V，则低电压复位电路会产生复位信号。

当在掩膜任选时选择了低电压复位功能，则 V_{DD}电压必须设置在 5 V±10%左右，如果太

低，例如降至 3.5 V，则会产生低电压复位。

二、复位状态

复位会对 CPU，I/O 端口等执行初始化，使有关部件处于某种初始状态，这些状态称为复位状态。下面介绍这些状态。

1. CPU 的复位状态

复位对 CPU 有如下影响：

(1) 把 $FF 装入堆栈指针；

(2) 对条件寄存器中的 I 位置“1”，屏蔽中断；

(3) 对中断状态及控制寄存器中的 IRQE 位置“1”，允许外部中断请求；

(4) 把用户定义的复位向量从地址 $03FE～$03FF 中取出，并送入程序计数器 PC 中；

(5) 对停止锁存器清“0”，允许 CPU 时钟工作；

(6) 对等待锁存器清“0”，把 CPU 从等待方式中唤醒。

2. I/O 端口寄存器的复位状态

复位对 I/O 端口寄存器有如下影响：

(1) 对数据方向寄存器 A 清“0”，故 PA 端口的所有引脚为输入方式；

(2) 对下拉寄存器 A 清“0”，接通 PA 口中的下拉晶体管(掩膜选择时选择下拉晶体管)；

(3) 对数据方向寄存器 B 清“0”，故 PB 口的两条引脚为输入方式；

(4) 对下拉寄存器 B 清“0”，接通 PB 口中的下拉晶体管(掩膜任选时选择下拉晶体管)；

(5) 对 PA 数据寄存器和 PB 数据寄存器无影响。

3. 多功能定时器的复位状态

复位对多功能定时器有如下影响：

(1) 对定时器状态及控制寄存器 TSCR 高 4 位清“0”，低 2 位置“1”，而其 D3，D2 不受影响；

(2) 对定时器计数器寄存器 TCNTR 清“0”。

4. COP Watchdog 的复位状态

复位时对 COP Watchdog 清“0”。掩膜任选时可选择 COP Watchdog。

2.5.3 节电方式

在 68HC05K0/K1 单片机中，有 4 种节电方式：停止方式、等待方式、暂停方式和数据保留方式。下面分别对这些方式进行介绍。

一、停止方式(STOP MODE)

STOP 指令使单片机进入 STOP 方式，即停止方式。停止方式是单片机的最低功耗方式。它对单片机有如下影响：

(1) 清 TOF，RTIF 标志，即清除所有悬挂着的定时器中断请求；

(2) 清 TOIE，RTIE 标志，屏蔽定时器中断；

(3) 对定时器计数器清“0”，即对 15 级脉冲计数器清“0”；

(4) 把中断状态及控制寄存器 ISCR 中的允许外部中断请求位 IRQE 置“1”，允许外部中断；

(5) 对条件码寄存器中的 I 位清“0”，允许中断；

(6) 停止内部振荡器，关断 CPU 时钟和定时器时钟，停止 COP Watchdog。

STOP 指令不会影响其他寄存器及 I/O 引脚的状态。

要使单片机退出停止方式，可采用下列方法。

方法 1　在$\overline{IRQ}$/V_{PP}引脚加上外部中断请求信号。

方法 2　在 PA 口的外部中断引脚上加上外部中断请求信号。

方法 3　在复位端$\overline{RESET}$上加上复位信号。

当单片机退出停止方式时，经过 4 064 个内部时钟周期后会处于稳定状态，并重新开始工作。

退出停止方式的定时时序如图 2-39 所示。

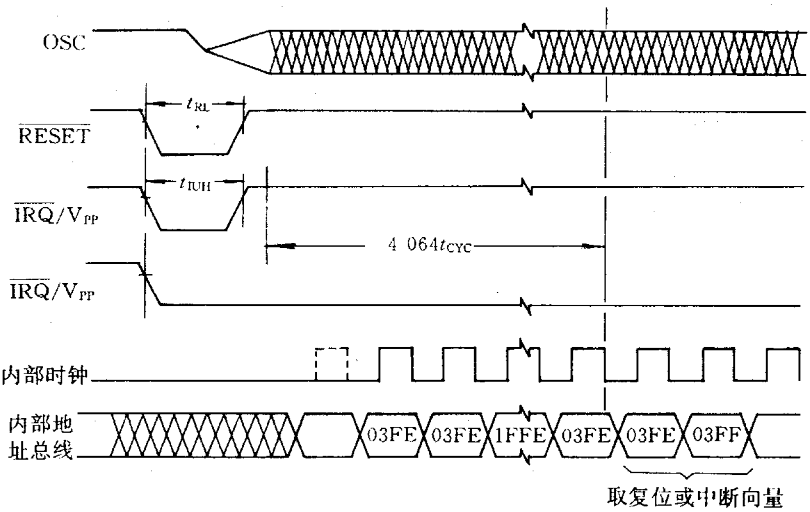


图 2-39　退出停止方式的定时时序

二、等待方式(WAIT MODE)

WAIT 指令使单片机进入较低功耗的工作方式，即等待方式。处于等待方式时，对单片机有如下影响：

(1) 对条件码寄存器中的 I 位清“0”，允许中断；

(2) 对中断状态及控制寄存器中的 IRQE 位置“1”，允许外部中断请求；

(3) 停止 CPU 时钟，但允许内部振荡器和定时器时钟继续工作。

WAIT 指令不影响其他寄存器或任何 I/O 引脚的状态。

要令单片机退出 WAIT 方式，可采用下列方法。

方法 1　在$\overline{IRQ}$/V_{PP}引脚加上外部中断请求信号。

方法 2　在 PA 口的外部中断引脚上加上外部中断请求信号。

方法 3　定时器中断，包括定时器溢出中断和实时中断。

方法 4　COP Watchdog 复位。

方法 5　外部复位。

三、暂停方式(HALT MODE)

如果在掩膜任选时屏蔽 STOP 指令，则 STOP 指令会使单片机进入暂停方式。

暂停方式和等待方式基本上相同，唯一的不同只是当单片机退出这种方式时需要 1～

4 064个内部时钟周期的恢复延时。

如果在掩膜任选时选择屏蔽 STOP 指令，则 COP Watchdog 不能由 STOP 指令关闭。

STOP，WAIT 和 HALT 方式的流程如图 2-40 所示。

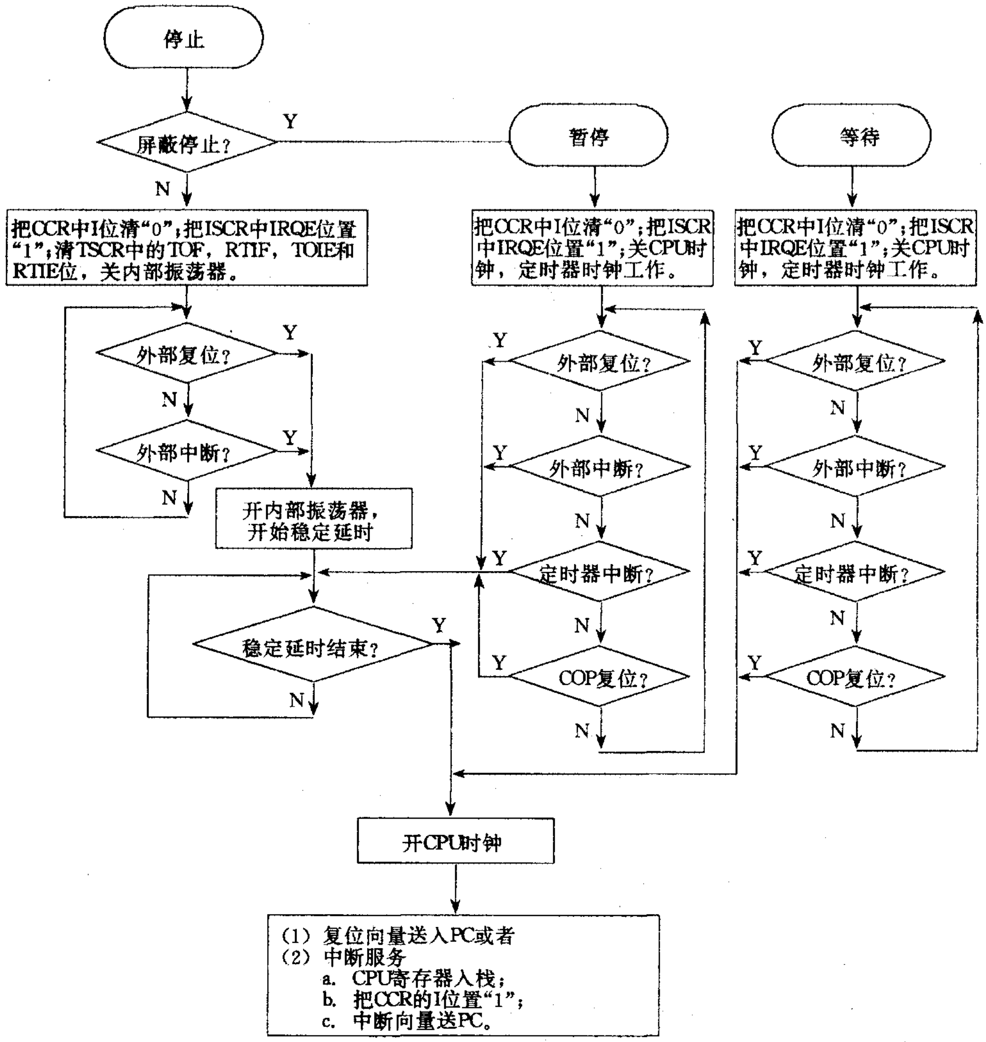


图2-40　STOP，WAIT和HALT方式流程

四、数据保留方式

当处于数据保留方式时，V_{DD}电压为 2 V，单片机保留 RAM 的内容和 CPU 寄存器的内容。

数据保留方式使单片机可以在低功耗的状态下保存数据，但这时 CPU 不能执行指令。

使单片机进入数据保留方式的方法如下：

(1) 首先在$\overline{\text{RESET}}$引脚加上逻辑“0”信号；

(2) 降低单片机的 V_{DD}电压，令其为 2 V。

这样，单片机就会进入数据保留方式。这种方式要求$\overline{\text{RESET}}$引脚一直保持逻辑“0”电平。

使单片机退出数据保留方式方法如下：

(1) 使单片机电压 V_{DD}恢复到正常工作电压，一般为 5 V；

(2) 使$\overline{RESET}$引脚电平恢复为逻辑“1”电平。

2.6 指令系统

68HC05K0/K1 的指令系统有 65 条基本指令，可形成 210 个不同的操作码，能表示 217 种操作意义，即有 217 条具体指令。

由于要用 210 个操作码表示 217 条指令，因此，有 7 条指令的操作码和另外 7 条指令的操作码是相同的。这 14 条指令如下：

指令	指令	操作码
ASL opr (DIR)	LSL opr(DIR)	38
ASL A (INH)	LSL A (INH)	48
ASL X (INH)	LSL X (INH)	58
ASL opr (IX1)	LSL opr (IX1)	68
ASL opr (IX)	LSL opr (IX)	78
ASL rel (REL)	BHS rel (REL)	24
BCS rel (REL)	BLO rel (REL)	25

当然，这些指令的名称不同，但具有相同操作码的指令在本质上是执行一样的动作。

2.6.1 寻址方式

为了灵活存取数据，CPU 有 8 种寻址方式。这些寻址方式决定了 CPU 寻找执行指令所需数据的方法。这 8 种寻址方式如下：

内在寻址(INH)

立即寻址(IMM)

直接寻址(DIR)

扩展寻址(EXT)

无偏置量变址寻址(IX)

8 位偏置量变址寻址(IX1)

16 位偏置量变址寻址(IX2)

相对寻址(REL)

下面分别介绍这些寻址方式及其意义。

一、内在寻址(INH)

内在寻址指令是那些没有操作数的指令，它无需存储器数据，并且是单字节指令。

这类指令只是执行内在的数据操作。例如，对 CPU 寄存器中的数据执行操作，最典型的是累加器加 1 指令 INCA；又例如，中断返回指令 RTI、停止指令 STOP、置进位位标志 SEC 等。

内在寻址指令如表 2-8 所示。

二、立即寻址(IMM)

立即寻址就是在指令中直接给出操作数。这种指令是双字节指令，第一个字节是操作码，

第二个字节是参与操作的立即数。立即寻址指令通常把立即数和累加器内容或变址寄存器内进行操作。

表 2-8 采用内在寻址方式的指令

指　　令	助记符	指　　令	助记符
算术左移	ASLA,ASLX	带进位左循环	ROLA,ROLX
算术右移	ASRA,ASRX	带进位右循环	RORA,RORX
清进位	CLC	栈指针复位	RSP
清中断屏蔽	CLI	从中断返回	RTI
清除	CLRA,CLRX	从子程序返回	RTS
求补	COMA,COMX	置进位	SEC
减 1	DECA,DECX	置中断屏蔽	SEI
加 1	INCA,INCX	停止	STOP
逻辑左移	LSLA,LSLX	软件中断	SWI
逻辑右移	LSRA,LSRX	把 A 送 X	TAX
A 乘 X	MUL	负或零测试	TSTA,TSTX
求反	NEGA,NEGX	把 X 送 A	TXA
空操作	NOP	等待	WAIT

立即寻址的指令如表 2-9 中所示。

表 2-9 采用立即寻址方式的指令

指　　令	助记符	指　　令	助记符
立即数连进位加 A	ADC	立即数异或 A	EOR
立即数加 A	ADD	立即数送 A	LDA
立即数“与”A	AND	立即数送 X	LDX
立即数与 A 逻辑比较	BIT	立即数“同或”A	ORA
立即数与 A 算术比较	CMP	A 减立即数和进位	SBC
立即数与 X 算术比较	CPX	A 减立即数	SUB

三、直接寻址(DIR)

直接寻址的指令字长 2 字节,它可以对存储器的首部 256 个地址进行访问。指令的第一个字节是操作码,第二个字节是操作数地址。在直接寻址中,CPU 自动把 \$00 作为操作数的高位地址,第二个字节作为操作数的低位地址,其寻址范围在 \$0000～\$00FF。

BRSET 和 BRCLR 是 2 条很特别的 3 字节指令,它们用直接寻址方式取得操作数,用相对寻址方式指明转移目的地址。

表 2-10 中给出了直接寻址有关指令。

表 2-10 直接寻址的指令

指　　令	助记符	指　　令	助记符
加内存、进位到 A	ADC	加 1	INC
加内存到 A	ADD	转移	JMP
内存"与"A	AND	转子程序	JSR
算术左移	ASL	内存送 A	LDA
算术右移	ASR	内存送 X	LDX
位清除	BCLR	逻辑左移	LSL
内存和 A 位测试	BIT	逻辑右移	LSR
位清除转移	BRCLR	求反	NEG
置位转移	BRSET	内存"同或"A	ORA
置位	BSET	通过进位循环左移	ROL
清除	CLR	通过进位循环右移	ROR
内存与 A 算术比较	CMP	A 减内存、进位	SBC
求补	COM	A 送内存	STA
内存与 X 算术比较	CPX	X 送内存	STX
减 1	DEC	A 减内存	SUB
内存"异或"A	EOR	负或零测试	TST

四、扩展寻址(EXT)

扩展寻址的指令是三字节指令，它可以访问存储器中的任何一个地址。在这种指令中，第一个字节是操作码，第二个字节是操作数的高字节地址，第三个字节是低字节地址。

所谓扩展寻址，是指相对于直接寻址方式，它的寻址范围广得多。

扩展寻址方式的指令如表 2-11 中所示。

表 2-11 扩展寻址方式的指令

指　　令	助记符	指　　令	助记符
加内存、进位到 A	ADC	转子程序	JSR
加内存到 A	ADD	内存送 A	LDA
内存"与"A	AND	内存送 X	LDX
内存与 A 位测试	BIT	内存"同或"A	ORA
内存与 A 算术比较	CMP	A 减内存、进位	SBC
内存与 X 算术比较	CPX	A 送内存	STA
内存"异或"A	EOR	X 送内存	STX
转移	JMP	A 减内存	SUB

五、无偏置量变址(IX)

无偏置量变址的指令是单字节指令。在这种指令中，变址寄存器 X 中的内容是操作数的低字节地址，而 CPU 自动用 \$00 作为高字节地址，所以无偏置量变址指令可以对 \$0000～\$00FF 地址进行寻址。

无偏置量变址指令通常用于在数据表中移动数据指针，或者用于保存频繁使用的 RAM 或 I/O 单元的地址。

表 2-12 给出了变址寻址的指令。

表 2-12　变址寻址指令

指　　令	助记符	无偏置量	8 位偏置量	16 位偏置量
加内存、进位到 A	ADC	√	√	√
加内存到 A	ADD	√	√	√
内存“与”A	AND	√	√	√
算术左移	ASL	√	√	
算术右移	ASR	√	√	
内存与 A 位测试	BIT	√	√	√
清除	CLR	√	√	
内存与 A 算术比较	CMP	√	√	√
求补	COM	√	√	
内存与 X 算术比较	CPX	√	√	√
减 1	DEC	√	√	
内存“异或”A	EOR	√	√	√
加 1	INC	√	√	
转移	JMP	√	√	√
转子程序	JSR	√	√	√
内存送 A	LDA	√	√	√
内存送 X	LDX	√	√	√
逻辑左移	LSL	√	√	
逻辑右移	LSR	√	√	
求反	NEG	√	√	
内存“同或”A	ORA	√	√	√
过进位循环左移	ROL	√	√	
过进位循环右移	ROR	√	√	
A 减内存、进位	SBC	√	√	√
A 送内存	STA	√	√	√
X 送内存	STX	√	√	√
A 减内存	SUB	√	√	√
负或零测试	TST	√	√	

六、8 位偏置量变址(IX1)

8 位偏置量变址是双字节指令，它可以在内存首部的 511 个地址中访问有关数据。CPU 把变址寄存器 X 的内容和指令第二个字节内容相加，其和便是操作数地址。8 位偏置量变址可以存取 $ 000～$ 01FE 地址范围中的数据。

8 位偏置量变址指令如表 2-12 中所示。

这种变址指令用于有 *n* 个数据的表格中选择第 *k* 个数据。数据表的表头在首部 256 个地址的任何位置都可以，表尾最多可到达地址 $ 01FE。一般而言，*k* 存放在变址寄存器 X 中，而数据表的首地址则用指令的第二个字节表示。

七、16 位偏置量变址(IX2)

这种变址方式的指令是三字节指令，它可以在存储器的任何地址存取数据。CPU 把变址寄存器 X 的内容和指令的第二、三个字节内容相加，得到的和是操作地址。在第二、三个字节中，第二个字节是高位地址，第三个字节是低位地址。

16 位编置量变址指令如表 2-12 中所示。

16 位偏置量变址指令用于在存储器任何地址中的 *n* 个数据的表格中选择第 *k* 个数据。当然，表格的首地址由指令的第二、三个字节给出，第 *k* 个数据的指针 k 则存放在变址寄存器 X 中。

八、相对变址(REL)

相对变址只用于转移指令。当转移条件满足时，CPU 就把指令中的偏置量和程序计数器 PC 中的内容相加，得出转移的目的地址。如果转移条件不满足，则 CPU 执行下一条指令。

转移指令中的偏置量是带符号的，而且用一个字节表示，故而转移指令中的偏置量为 －127～＋128。偏置量用 2 的补码表示。转移指令的转移范围一般是转移指令的下一条指令地址的－127～＋128 之间的地址。

相对变址指令如表 2-13 中所示。

表 2-13　相对变址寻址指令

指　　令	助记符	指　　令	助记符
进位为 0 转移	BCC	I=0 转移	BMC
进位为 1 转移	BCS	负转移	BMI
相等转移	BEQ	I=1 转移	BMS
半进位为 0 转移	BHCC	不等转移	BNE
半进位为 1 转移	BHCS	正转移	BPL
较高转移	BHI	常转移	BRA
较高或相等转移	BHS	位为 0 转移	BRCLR
$\overline{IRQ}$为高转移	BIH	位为 1 转移	BRSET
$\overline{IRQ}$为低转移	BIL	不转移	BRN
较低转移	BLO	转向子程序	BSR
较低或相等转移	BLS		

2.6.2 指令类型

在 68HC05K0/K1 的指令系统中，指令被分成 5 种类型：

(1) 寄存器/存储器类型；

(2) 读—修改—写类型；

(3) 转移类型；

(4) 位操作类型；

(5) 控制类型。

这一部分将分别对各类型的指令进行介绍。

一、寄存器/存储器类指令

这种类型指令的大多数含有 2 个操作数，其中一个操作数在累加器 A 或变址寄存器中，另一个操作数在存储器中。因此，它们称为寄存器/存储器指令。大多数这种类型的指令采用下列寻址方式。

(1) 立即寻址(IMM)；

(2) 直接寻址(DIR)；

(3) 扩展寻址(EXT)；

(4) 无偏置量变址寻址(IX)；

(5) 8 位偏置量变址寻址(IX1)；

(6) 16 位偏置量变址寻址(IX2)。

寄存器/存储器类指令如表 2-14 所示。

表 2-14 寄存器/存储器类指令

指　　令	助记符	指　　令	助记符
内存送 A	LDA	A 减内存、进位	SBC
内存送 X	LDX	内存“与”A	AND
A 送内存	STA	内存“同或”A	ORA
X 送内存	STX	A 与内存算术比较	CMP
内存加 A	ADD	X 与内存算术比较	CPX
内存、进位加 A	ADC	内存与 A 位测试	BIT
A 减内存	SUB	A 乘 X	MUL

二、读—修改—写类指令

所谓读—修改—写类指令，是指能读存储器或寄存器的内容，然后把这些内容加以修改，最后又把修改后的内容写回存储器或寄存器中的指令。

读—修改—写类指令采用以下 4 种寻址方式：

(1) 内在寻址(INH)；

(2) 直接寻址(DIR)；

(3) 无偏置量变址寻址(IX)；

(4) 8 位偏置量变址寻址(IX1)。

在读—修改—写类指令中还包括负或零测试指令 TST。这条指令较为特殊,因为它并不把读出的内容修改后写回存储器或寄存器中。

读—修改—写类指令如表 2-15 所示。

表 2-15　读—修改—写类指令

指　　　令	助记符
加 1	INC
减 1	DEC
清除	CLR
求补	COM
求反	NEG
过进位循环左移	ROL
过进位循环右移	ROR
逻辑左移	LSL
逻辑右移	LSR
算术右移	ASR
负或零测试	TST

三、转移类指令

转移类指令允许 CPU 暂时中断程序执行的原有顺序,而转到别的地方去执行。一般的转移指令需要寄存器操作数,而无条件转移指令 JMP 和转子程序指令 JSR 则无需寄存器操作数。这两种转移指令用下列寻址方式:

(1) 直接寻址(DIR);

(2) 扩展寻址(EXT);

(3) 无偏置量变址寻址(IX);

(4) 8 位偏置量变址寻址(IX1);

(5) 16 位偏置量变地寻址(IX2)。

条件转移指令和无条件转移指令执行的情况相似,所不同的是条件转移指令只在条件满足时产生转移,条件不满足时不产生转移。所有的条件转移指令都采用相对寻址方式。

位测试转移指令的转移条件取决于存储器首部 256 个存储单元任何可读位的状态。这种位测试转移指令是三字节指令,它采用直接寻址和相对寻址两种寻址方式。第一个字节是指令的操作码,第二个字节是被测试的字节的直接地址,第三个字节是带符号的偏置量。转移地址是由下条指令的地址加上带符号的偏置量形成的,故转移地址是相对寻址的。

转移类指令如表 2-16 中所示。

四、位操作类指令

CPU 可以在存储器首部 256 个地址任何可写位执行置“1”或清“0”操作。端口寄存器、端口数据方向寄存器、定时器寄存器和片内 RAM 都位于存储器首部 256 个地址之内。位操作类

指令采用直接寻址方式。

表 2-16 转移类指令

指　　令	助记符	指　　令	助记符
常转移	BRA	清半进位位转移	BHCC
不转移	BRN	置半进位位转移	BHCS
清位转移	BRCLR	正转移	BPL
置位转移	BRSET	负转移	BMI
较高转移	BHI	清 I 转移	BMC
较低或相等转移	BLS	置 I 转移	BMS
清进位转移	BCC	中断引脚低转移	BIL
较高或相等转移	BHS	中断引脚高转移	BIH
置进位转移	BCS	转子程序	BSR
较低转移	BLO	转移	JMP
不等转移	BND	转子程序	JSR
相等转移	BEQ		

位操作类指令如表 2-17 所示。

表 2-17 位操作类指令

指　　令	助记符
位置“1”	BSET
位清“0”	BCLR
清位转移	BRCLR
置位转移	BRSET

五、控制类指令

改变有关寄存器或标志的内容，从而控制 CPU 的操作，这样的指令为控制类指令。控制类指令如表 2-18 中所示，它们采用内部寻址方式。

表 2-18 控制类指令

指　　令	助记符	指　　令	助记符
A 送 X	TAX	软件中断	SWI
X 送 A	TXA	从子程序返回	RTI
进位位置“1”	SEC	栈指针清“0”	RSP
进位位清“0”	CLC	空操作	NOP
对 I 置“1”	SEI	停止	STOP
对 I 清“0”	CLI	等待	WAIT

2.6.3 指令系统及指令说明

68HC05K0/K1 的指令系统说明如表 2-19 和表 2-20 中所示。

表 2-19 中给出了指令操作码，操作数的结构，工作周期，标志变化情况，寻址方式和操作意义。

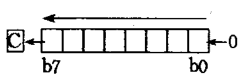
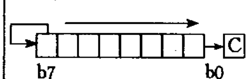
表 2-20 中给出了操作码图。

在表 2-19 及表 2-20 中有一些符号及其简写，为了便于了解指令系统，这里先对这些符号及简写加以说明。

A	累加器
C	进位/借位
CCR	条件码寄存器
dd	直接寻址中的操作数地址
dd rr	操作数地址(dd)和偏置量(rr)
DIR	直接寻址
ee ff	偏置量的高节字(ee)及低字节(ff)
EXT	扩展寻址
ff	8 位偏置量
H	半进位位
hh ll	操作数的地址高位(hh)和低位(ll)
I	中断标志
ii	立即寻址中的操作数字节
IMM	立即寻址方式
INH	内在寻址方式
IX	无偏置量变址寻址方式
IX1	8 位偏置量变址寻址方式
IX2	16 位偏置量变址寻址方式
M	存储器单元，1 个字节
N	负标志
n	位编号(n=1,2,3,…,7)
opr	操作数字节
PC	程序计数器
PCH	程序计数器高字节
PCL	程序计数器低字节
REL	相对寻址方式
rel	相对寻址的偏置字节
rr	转移指令的偏置字节
SP	堆栈指针
X	变址寄存器
Z	零标志
*	逻辑"与"
-	无影响
?	如果
—	非，不
()	内容
←	装入

:	联系、连接
×	乘
−()	求反
+	“同或”
↕	如果为真，则置“1”；如果不为真则清“0”。
⊕	“异或”
+	相加
−	相减

表 2-19　指令系统表

指令形式	操作	说明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
ADC opr	带进位加	A←(A)+(M)+C	IMM	A9	ii	2	↕	-	↕	↕	↕
			DIR	B9	dd	3					
			EXT	C9	hh ll	4					
			IX2	D9	ee ff	5					
			IX1	E9	ff	4					
			IX	F9		3					
ADD opr	不带进位加	A←(A)+(M)	IMM	AB	ii	2	↕	-	↕	↕	↕
			DIR	BB	dd	3					
			EXT	CB	hh ll	4					
			IX2	DB	ee ff	5					
			IX1	EB	ff	4					
			IX	FB		3					
AND opr	逻辑“与”	A←(A)·(M)	IMM	A4	ii	2	↕	-	↕	↕	-
			DIR	B4	dd	3					
			EXT	C4	hh ll	4					
			IX2	D4	ee ff	5					
			IX1	E4	ff	4					
			IX	F4		3					
ASL opr	算术左移		DIR	38	dd	5	-	-	↕	↕	↕
ASLA			INH	48		3					
ASLX			INH	58		3					
ASL opr			IX1	68	ff	6					
ASL opr			IX	78		5					
ASR opr	算术右移		DIR	37	dd	5	-	-	↕	↕	↕
ASRA			INH	47		3					
ASRX			INH	57		3					
ASR opr			IX1	67	ff	6					
ASR opr			IX	77		5					

续表 2-19

指令形式	操　作	说　明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
BCC rel	进位位清"0"转移	? C=0	REL	24	rr	3	-	-	-	-	-
BCLR n opr	第 n 位清"0"	Mn←0	DIR(b0)	11	dd	5	-	-	-	-	-
			DIR(b1)	13	dd	5					
			DIR(b2)	15	dd	5					
			DIR(b3)	17	dd	5					
			DIR(b4)	19	dd	5					
			DIR(b5)	1B	dd	5					
			DIR(b6)	1D	dd	5					
			DIR(b7)	1F	dd	5					
BCS rel	进位位置"1"转移	? C=1	REL	25	rr	3	-	-	-	-	-
BEQ rel	相等转移	? Z=1	REL	27	rr	3	-	-	-	-	-
BHCC rel	半进位位清"0"转移	? H=0	REL	28	rr	3	-	-	-	-	-
BHCS rel	半进位位置"1"转移	? H=1	REL	29	rr	3	-	-	-	-	-
BHI rel	较高转移	? C+Z=0	REL	22	rr	3	-	-	-	-	-
BHS rel	较高或相等转移	? C=0	REL	24	rr	3	-	-	-	-	-
BIH rel	$\overline{IRQ}$引脚高转移	? $\overline{IRQ}$=1	REL	2F	rr	3	-	-	-	-	-
BIL rel	$\overline{IRQ}$引脚低转移	? $\overline{IRQ}$=0	REL	2E	rr	3	-	-	-	-	-
BIT rel	内存与 A 位测试	(A)·(M)	IMM	A5	ii	2	-	-	↕	↕	-
			DIR	B5	dd	3					
			EXT	C5	hh ll	4					
			IX2	D5	ee ff	5					
			IX1	E5	ff	4					
			IX	F5		3					
BLO rel	较低转移	? C=1	REL	25	rr	3	-	-	-	-	-
BLS rel	较低或相等转移	? C+Z=1	REL	23	rr	3	-	-	-	-	-
BMC rel	I=0 转移	? I=0	REL	2C	rr	3	-	-	-	-	-
BMI rel	负转移	? N=1	REL	2B	rr	3	-	-	-	-	-
BMS rel	I=1 转移	? I=1	REL	2D	rr	3	-	-	-	-	-

指令形式	操　作	说　明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
BNE rel	不相等转移	? Z=0	REL	26	rr	3	-	-	-	-	-
BPL rel	正转移	? N=0	REL	2A	rr	3	-	-	-	-	-
BRA rel	常转移	? I=1	REL	20	rr	3	-	-	-	-	-
BRCLR *n* opr rel	第 *n* 位为“0”转移	? M*n*=0	DIR(b0)	01	dd rr	5	-	-	-	-	↕
			DIR(b1)	03	dd rr	5					
			DIR(b2)	05	dd rr	5					
			DIR(b3)	07	dd rr	5					
			DIR(b4)	09	dd rr	5					
			DIR(b5)	0B	dd rr	5					
			DIR(b6)	0D	dd rr	5					
			DIR(b7)	0F	dd rr	5					
BRN rel	不转移	? I=0	REL	21	rr	3	-	-	-	-	-
BRSET *n* opr rel	第 *n* 位为“1”转移	? M*n*=1	DIR(b0)	00	dd rr	5	-	-	-	-	↕
			DIR(b1)	02	dd rr	5					
			DIR(b2)	04	dd rr	5					
			DIR(b3)	06	dd rr	5					
			DIR(b4)	08	dd rr	5					
			DIR(b5)	0A	dd rr	5					
			DIR(b6)	0C	dd rr	5					
			DIR(b7)	0E	dd rr	5					
BSET *n* opr	第 *n* 位置“1”	M*n*←1	DIR(b0)	10	dd	5	-	-	-	-	-
			DIR(b1)	12	dd	5					
			DIR(b2)	14	dd	5					
			DIR(b3)	16	dd	5					
			DIR(b4)	18	dd	5					
			DIR(b5)	1A	dd	5					
			DIR(b6)	1C	dd	5					
			DIR(b7)	1E	dd	5					
BSR rel	转子程序	PC←(PC)+2;push(PCL) SP←(SP)−1;push(PCH) SP←(SP)−1 PC←(PC)+rel	REL	AD	rr	6	-	-	-	-	-

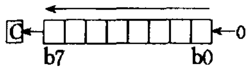
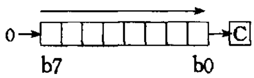
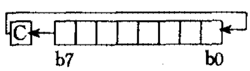
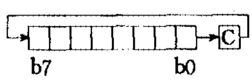
续表 2-19

指令形式	操作	说明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
CLC	清进位位	C←0	INH	98		2	-	-	-	-	0
CLI	清中断标志	I←0	INH	9A		2	-	0	-	-	-
CLR opr	寄存器清"0"	M←$00	DIR	3F	dd	5	-	-	0	1	-
CLRA		A←$00	INH	4F		3					
CLRX		X←$00	INH	5F		3					
CLR opr		M←$00	IX1	6F	ff	6					
CLR opr		M←$00	IX	7F		5					
CMP opr	内存与 A 比较	(A)−(M)	IMM	A1	ii	2	-	-	↕	↕	↕
			DIR	B1	dd	3					
			EXT	C1	hh ll	4					
			IX2	D1	ee ff	5					
			IX1	E1	ff	4					
			IX	F1		3					
COM opr	对寄存器内容求补	$M \leftarrow \overline{M} = \$FF-(M)$	DIR	33	dd	5	-	-	↕	↕	1
COMA		$A \leftarrow \overline{A} = \$FF-(A)$	INH	43		3					
COMX		$X \leftarrow \overline{X} = \$FF-(X)$	INH	53		3					
COM opr		$M \leftarrow \overline{M} = \$FF-(M)$	IX1	63	ff	6					
COM opr		$M \leftarrow \overline{M} = \$FF-(M)$	IX	73		5					
CPX opr	内存与 X 比较	(X)−(M)	IMM	A3	ii	2	-	-	↕	↕	↕
			DIR	B3	dd	3					
			EXT	C3	hh ll	4					
			IX2	D3	ee ff	5					
			IX1	E3	ff	4					
			IX	F3		3					
DEC opr	内存或寄存器内容减 1	M←(M)−1	DIR	3A	dd	5	-	-	↕	↕	-
DECA		A←(A)−1	INH	4A		3					
DECX		X←(X)−1	INH	5A		3					
DEC opr		M←(M)−1	IX1	6A	ff	6					
DEC opr		M←(M)−1	IX	7A		5					

续表 2-19

指令形式	操　作	说　明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
EOR opr	内存“异或”A	A←(A)⊕(M)	IMM	A8	ii	2	-	-	↕	↕	-
			DIR	B8	dd	3					
			EXT	C8	hh ll	4					
			IX2	D8	ee ff	5					
			IX1	E8	ff	4					
			IX	F8		3					
INC opr	内存或寄存器内容加 1	M←(M)+1	DIR	3C	dd	5	-	-	↕	↕	-
INCA		A←(A)+1	INH	4C		3					
INCX		X←(X)+1	IX2	5C	ff	3					
INC opr		M←(M)+1	IX1	6C		6					
INC opr		M←(M)+1	IX	7C		5					
JMP opr	无条件跳转	PC←jump address	DIR	BC	dd	2	-	-	-	-	-
			EXT	CC	hh ll	3					
			IX2	DC	ee ff	4					
			IX1	EC	ff	3					
			IX	FC		2					
JSR opr	转子程序	PC←(PC)+n(n=1,2,3)	DIR	BD	dd	5	-	-	-	-	-
		Push(PCL);SP←(SP)−1	EXT	CD	hh ll	6					
		Push(PCH);SP←(SP)−1	IX2	DD	ee ff	7					
		PC←conditional address	IX1	ED	ff	6					
			IX	FD		5					
LDA opr	内存送 A	A←(M)	IMM	A6	ii	2	-	-	↕	↕	-
			DIR	B6	dd	3					
			EXT	C6	hh ll	4					
			IX2	D6	ee ff	5					
			IX1	E6	ff	4					
			IX	F6		3					
LDX opr	内存送 X	X←(M)	IMM	AE	ii	2	-	-	↕	↕	-
			DIR	BE	dd	3					
			EXT	CE	hh ll	4					
			IX2	DE	ee ff	5					
			IX1	EE	ff	4					
			IX	FE		3					

续表 2-19

指令形式	操作	说明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
LSL opr	逻辑左移		DIR	38	dd	5	-	-	↕	↕	↕
LSLA			INH	48		3					
LSLX			INH	58		3					
LSL opr			IX1	68	ff	6					
LSL opr			IX	78		5					
LSR opr	逻辑右移		DIR	34	dd	5	-	-	0	↕	↕
LSRA			INH	44		3					
LSRX			INH	54		3					
LSR opr			IX1	64	ff	6					
LSR opr			IX	74		5					
MUL	无符号乘	X:A←(X)×(A)	INH	42		11	0	-	-	-	0
NEG opr	对内存或寄存器内容求反	M←-(M)=＄00-(M)	DIR	30	dd	5	-	-	↕	↕	↕
NEGA		A←-(A)=＄00-(A)	INH	40		3					
NEGX		X←-(X)=＄00-(X)	INH	50		3					
NEG opr		M←-(M)=＄00-(M)	IX1	60	ff	6					
NEG opr		M←-(M)=＄00-(M)	IX	70		5					
NOP	空操作		INH	9D		2	-	-	-	-	-
ORA opr	内存“同或”A	A←(A)+(M)	IMM	AA	ii	2	-	-	↕	↕	-
			DIR	BA	dd	3					
			EXT	CA	hh ll	4					
			IX2	DA	ee ff	5					
			IX1	EA	ff	4					
			IX	FA		3					
ROL opr	过进位位循环左移		DIR	39	dd	5	-	-	↕	↕	↕
ROLA			INH	49		3					
ROLX			INH	59		3					
ROL opr			IX1	69	ff	6					
ROL opr			IX	79		5					
ROR opr	过进位位循环右移		DIR	36	dd	5	-	-	↕	↕	↕
RORA			INH	46		3					
RORX			INH	56		3					
ROR opr			IX1	66	ff	6					
ROR opr			IX	76		5					

续表 2-19

指令形式	操　作	说　明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
RSP	复位堆栈指针	SP←$00FF	INH	9C		2	From Stack				
RTI	中断返回	SP←(SP)+1;pull(CCR) SP←(SP)+1;pull(A) SP←(SP)+1;pull(X) SP←(SP)+1;pull(PCH) SP←(SP)+1;pull(PCL)	INH	80		9	↕	↕	↕	↕	↕
RTS	从子程序返回	SP←(SP)+1;pull(PCH) SP←(SP)+1;pull(PCL)	INH	81		6	-	-	-	-	-
SBC opr	A 减内存、进位	A←(A)−(M)−C	IMM DIR EXT IX2 IX1 IX	A2 B2 C2 D2 E2 F2	ii dd hh ll ee ff ff	2 3 4 5 4 3	-	-	↕	↕	↕
SEC	置进位位	C←1	INH	99		2	-	-	-	-	1
SEI	置中断标志	I←1	INH	9B		2	-	1	-	-	-
STA opr	A 送内存	M←(A)	DIR EXT IX2 IX1 IX	B7 C7 D7 E7 F7	dd hh ll ee ff ff	4 5 6 5 4	-	-	↕	↕	-
STOP	停止		INH	8E		2	-	0	-	-	-
STX opr	X 送内存	M←(X)	DIR EXT IX2 IX1 IX	BF CF DF EF FF	dd hh ll ee ff ff	4 5 6 5 4	-	-	↕	↕	-
SUB opr	A 减内存	A←(A)−(M)	IMM DIR EXT IX2 IX1 IX	A0 B0 C0 D0 E0 F0	ii dd hh ll ee ff ff	2 3 4 5 4 3	-	-	↕	↕	↕

续表 2-19

指令形式	操　作	说　明	寻址方式	机器码		周期	条件码				
				操作码	操作数		H	I	N	Z	C
SWI	软件中断	PC←(PC)+1;push(PCL) SP←(SP)−1;push(PCH) SP←(SP)−1;push(X) SP←(SP)−1;push(A) SP←(SP)−1;push(CCR) SP←(SP)−1;I←1 PCH←中断向量高位字节 PCL←中断向量低位字节	INH	83		10	-	1	-	-	-
TAX	A 送 X	X←(A)	INH	97		2	-	-	-	-	-
TST opr TSTA TSTX TST opr TST opr	负或零测试	(M)−$00	DIR INH INH IX1 IX	3D 4D 5D 6D 7D	dd ff 	4 3 3 5 4	-	-	↕	↕	-
TXA	X 送 A	A←(X)	INH	9F		2	-	-	-	-	-
WAIT	等待		INH	8F		2	-	0	-	-	-

表 2-20　68HC05 操作码图

	位操作		转　移	读			写		控　制		寄存器/存储器						
	DIR	DIR	REL	DIR	INH	INH	IX1	IX	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
HI LO	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	HI LO
0 0000	5 BRSET0 3 DIR	5 BSET0 2 DIR	3 BRA 2 REL	5 NEG 2 DIR	3 NEGA 1 INH	3 NEGX 1 INH	6 NEG 2 IX1	5 NEG 1 IX	9 RTI 1 INH		2 SUB 2 IMM	3 SUB 2 DIR	4 SUB 3 EXT	5 SUB 3 IX2	4 SUB 2 IX1	3 SUB 1 IX	0 0000
1 0001	5 BRCLR0 3 DIR	5 BCLR0 2 DIR	3 BRN 2 REL						6 RTS 1 INH		2 CMP 2 IMM	3 CMP 2 DIR	4 CMP 3 EXT	5 CMP 3 EXT	4 CMP 2 IX1	3 CMP 1 IX	1 0001
2 0010	5 BRSET1 3 DIR	5 BSET1 2 DIR	3 BHI 2 REL		11 MUL 1 INH						2 SBC 2 IMM	3 SBC 2 DIR	4 SBC 3 EXT	5 SBC 3 IX2	4 SBC 2 IX1	3 SBC 1 IX	2 0010
3 0011	5 BRCLR1 3 DIR	5 BSCLR1 2 DIR	3 BLS 2 REL	5 COM 2 DIR	3 COMA 1 INH	3 COMX 1 INH	6 COM 2 IX1	5 COM 1 IX	10 SWI 1 INH		2 CPX 2 IMM	3 CPX 2 DIR	4 CPX 3 EXT	5 CPX 3 IX2	4 CPX 2 IX1	3 CPX 1 IX	3 0011
4 0100	5 BRSET2 3 DIR	5 BSET2 2 DIR	3 BCC 2 REL	5 LSR 2 DIR	3 LSRA 1 INH	3 LSRX 1 INH	6 LSR 2 IX1	5 LSR 1 IX			2 AND 2 IMM	3 AND 2 DIR	4 AND 3 EXT	5 AND 3 IX2	4 AND 2 IX1	3 AND 1 IX	4 0100
5 0101	5 BRCLR2 3 DIR	5 BCLR2 2 DIR	3 BCS 2 REL								2 BIT 2 IMM	3 BIT 2 DIR	4 BIT 3 EXT	5 BIT 3 IX2	4 BIT 2 IX1	3 BIT 1 IX	5 1010
6 0110	5 BRSET3 3 DIR	5 BSET3 2 DIR	3 BNE 2 REL	5 ROR 2 DIR	3 RORA 1 INH	3 RORX 1 INH	6 ROR 2 IX1	5 ROR 1 IX			2 LDA 2 IMM	3 LDA 2 DIR	4 LDA 3 EXT	5 LDA 3 IX2	4 LDA 2 IX1	3 LDA 1 IX	6 0110
7 0111	5 BRCLR3 3 DIR	5 BCLR3 2 DIR	3 BEQ 2 REL	5 ASR 2 DIR	3 ASRA 1 INH	3 ASRX 1 INH	6 ASR 2 IX1	5 ASR 1 IX		2 TAX 1 INH		4 STA 2 DIR	5 STA 3 EXT	6 STA 3 IX2	5 STA 3 IX1	4 STA 1 IX	7 0111

	位操作		转 移	读		写			控 制		寄存器/存储器						
	DIR	DIR	REL	DIR	INH	INH	IX1	IX	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
HI LO	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	HI LO
8 1000	5 BRSET4 3 DIR	5 BSET4 2 DIR	3 BHCC 2 REL	5 LSL 2 DIR	3 LSLA 1 INH	3 LSLX 1 INH	6 LSL 2 IX1	5 LSL 1 IX		2 CLC 1 INH	2 EOR 2 IMM	3 EOR 2 DIR	4 EOR 3 EXT	5 EOR 3 IX2	4 EOR 2 IX1	3 EOR 1 IX	8 1000
9 1001	5 BRCLR4 3 DIR	5 BRCLR4 2 DIR	3 BHCS 2 REL	5 ROL 2 DIR	3 ROLA 1 INH	5 ROLX 1 INH	6 ROL	5 ROL 1 IX		2 SEC 1 INH	2 ADC 2 IMM	3 ADC 2 DIR	4 ADC 3 EXT	5 ADC 3 IX2	4 ADC 2 IX1	3 ADC 1 IX	9 1001
A 1010	5 BRSET5 3 DIR	5 BSET5 2 DIR	3 BPL 2 REL	5 DEC 2 DIR	3 DECA 1 INH	3 DECX 1 INH	6 DEC 2 INH	5 DEC 1 IX		2 CLI 1 INH	2 ORA 2 IMM	3 ORA 2 DIR	4 ORA 3 EXT	5 ORA 3 IX2	4 ORA 2 IX1	5 ORA 1 IX	A 1010
B 1011	5 BRCLR5 3 DIR	5 BCLR5 2 DIR	3 BMI 2 REL							2 SEI 1 INH	2 ADD 2 IMM	3 ADD 2 DIR	4 ADD 3 EXT	5 ADD 3 IX2	4 ADD 2 IX1	3 ADD 1 IX	B 1011
C 1100	5 BRSET6 3 DIR	5 BSET6 2 DIR	3 BMC 2 REL	5 INC 2 DIR	3 INCA 1 INH	3 INCX 1 INH	6 INC 2 INH	5 INC 1 IX		2 RSP 1 INH		2 JMP 2 DIR	3 JMP 3 EXT	4 JMP 3 IX2	3 JMP 2 IX1	2 JMP 1 IX	C 1100
D 1101	5 BRCLR6 3 DIR	5 BCLR6 2 DIR	3 BMS 2 REL	4 TST 2 DIR	3 TSTA 1 INH	3 TSTX 1 INH	6 TST 2 INH	4 TST 1 IX		2 NOP 1 INH	6 BSR 2 REL	5 JSR 2 DIR	6 JSR 3 EXT	7 JSR 3 IX2	6 JSR 2 IX1	5 JSR 1 IX	D 1101
E 1110	5 BRSET7 3 DIR	5 BSET7 2 DIR	3 BIL 2 REL						2 STOP 1 INH		2 LDX 2 IMM	3 LDX 2 DIR	4 LDX 3 EXT	5 LDX 3 IX2	4 LDX 2 IX1	3 LDX 1 IX	E 1110
F 1111	5 BRCLR7 3 DIR	5 BCLR7 2 DIR	3 BIH 2 REL	5 CLR 2 DIR	3 CLRA 1 INH	3 CLRX 1 INH	6 CLR 2 INH	5 CLR 1 IX	2 WAIT 1 INH	2 TXA 1 INH		4 STX 2 DIR	5 STX 3 EXT	6 STX 3 IX2	5 STX 2 IX1	4 STX 1 IX	F 1111

寻址方式缩写

INH	Inherent	REL	Relative	IMM	Immediate	IX	Indexed, No Offset
DIR	Direct	IX1	Indexed, 8-Bit Offset	EXT	Extended	IX2	Indexed, 16-Bit Offset

第三章　MC68HC05J 系列单片机原理

MC68HC05J 系列单片机是一种低成本、高性能的 8 位单片机，包括 J1，J1A，J2，J3 等几种。该系列体积小，封装采用 20 只引脚的双列直插或双列表面封装。体积虽小，但其内部功能齐全，其中央处理器仍是 MC68HC05 大家族统一的高性能 CPU 系统，具备相同的指令系统，编程容易；内部具有完整的中断系统、多功能定时器、COP 监视器系统；内部程序存储空间最大可有 2 048 B，最小为 1 024 B，具有一定的程序存储能力。该系列因其低成本、高性能、体积小而得到广泛应用，尤其是在仪器、仪表、小家电控制器中常用到该系列。

3.1　MC68HC05J 系列单片机的结构

3.1.1　MC68HC05J 系列基本特性

J 系列包括 J1，J1A，J2，J3 等四个品种，这四个品种因内部 RAM 和 ROM 不同而有所区别，其他方面则大致相同。以下为 J 系列的共有特性。

(1) MC68HC05 系列中央处理器；

(2) 14 根双向输入/输出端口，且 J1A，J3 系列有的引脚的输出驱动能力可达到 8 mA 的灌入电流；

(3) 程序存储空间大小：1 040 B(J1 系列)，2 064 B(J2 系列)，1 240 B(J1A 系列)，2 048 B (J3 系列)；

(4) 数据存储空间大小：64 B(J1，J1A 系列)，112 B(J2 系列)，128 B(J3 系列)；

(5) 一个 15 级多功能定时器，J3 系列还外加一个 16 位多功能定时器，且具备输入捕捉，输出比较功能；

(6) 一个外部中断 $\overline{IRQ}$，另外，J3，J1A 系列还具有四个外部按键中断输入；

(7) 具有三种低功耗工作方式：停止方式(STOP)、等待方式(WAIT)以及数据保留方式(DATA_RETENTION)；

(8) 具有三种中断方式：外部中断、定时器中断和软中断；对于 J3 系列还具有 16 位定时器的输入捕捉、输出比较以及溢出等三种中断；

(9) 完整的复位系统，具有自动上电复位、外部复位、COP 复位、非法地址复位等多种复位方式；

(10) 多种振荡方式：晶振、*RC* 阻容振荡以及外部振荡，最大振荡频率为 4 MHz；

(11) 64 B 的堆栈。

3.1.2　MC68HC05J 系列基本结构

一、MC68HC05J 系列单片机的结构

该系列的单片机结构如图 3-1 所示。很明显，它是由 CPU、存储器、输入/输出端口、振荡器和定时器组成。

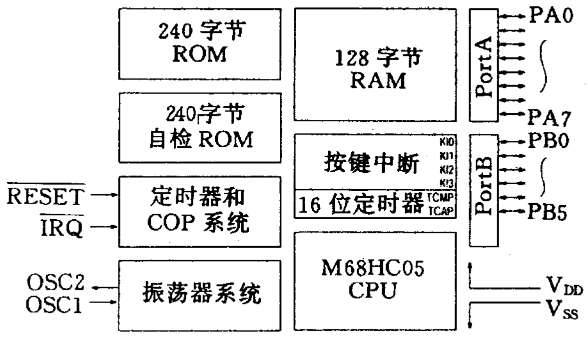


图 3-1　J 系列单片机结构框图

MC68HC05J 系列的 CPU 采用的是 MC68HC05 系列统一使用的中央处理器，由 CPU 控制电路、算术逻辑单元和 CPU 寄存器组成。其 CPU 寄存器同样为累加寄存器 A、变址寄存器 X、栈指针 SP、程序计数器 PC 以及条件码寄存器 CCR 等五种。

存储器由静态存储器 RAM，用户程序存储器 ROM 和自检程序 ROM 组成。用户 ROM 最大可达 2 064 B，最小为 1 040 B，数据存储器 RAM 最大达 128 B，最小为 64 B。

有 14 个双向输入/输出端口，其中 PA 端口 8 个，PB 端口 6 个。对于 J1A，J3 系列，其 PA 端口的驱动能力较大，可灌入 8 mA 电流；PA0，PA1，PA2，PA3 具有中断功能。

其定时器为 15 级多功能定时器，该定时器可方便地产生各种定时信号，包括 COP 电路计时信号。另外，J3 系列还有 16 位多功能定时器，具有输入捕捉、输出比较功能。

振荡器电路包含在单片机内部，只要外接振荡元件，便可产生振荡信号；产生的振荡信号经 2 分频后作为 CPU 操作的时钟信号。

二、MC68HC05J 系列引脚描述

MC68HC05J 系列单片机有 20 脚 DIP 封装和 20 脚 SDIP 封装两种形式，如图 3-2 所示。

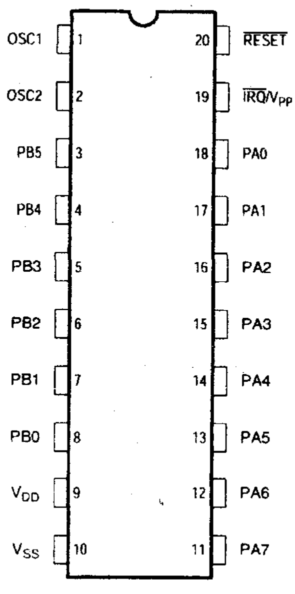


图 3-2　J 系列单片机引脚

1. V_{DD}与 V_{SS}

它们是单片机电源的输入端，V_{DD}接＋5 V 或＋3.3 V，V_{SS}接 0 V。具体应用时，需要在这两端尽量靠近单片机的地方连接一对电容，其中一个为 0.1 μF 的无感电容，另一个为容量较大的电解电容。无感电容可采用独石或钽电容，不可使用电解电容。这样可以降低单片机运行时的电源噪声，提高单片机运行的的稳定性。

2. OSC1 和 OSC2

它们是振荡器连接振荡元件的引脚。用户可以通过选择，决定使用何种振荡元件。

晶振和陶瓷振荡方式

当选用该方式时，要在 OSC1 与 OSC2 之间跨接晶体振荡元件或者陶瓷振荡元件。如图 3-3所示。电容与振荡元件要尽量靠近单片机的 OSC1，OSC2 两引脚，以减少振荡电路与其他

电路间的相互影响。电容采用无感电容，容量为 25 pF～27 pF。

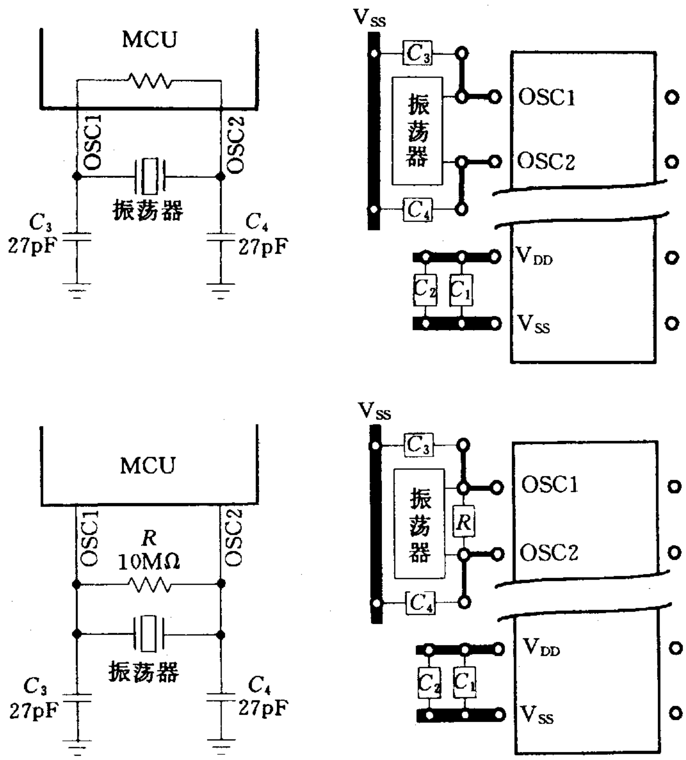


图 3-3　晶振

阻容振荡方式

在该种振荡方式下，只要在 OSC1 与 OSC2 间跨接一电阻即可。*RC* 阻容振荡方式是成本最低的振荡方式，但其精度及稳定性较差，工作频率不能太高。在该方式下，电阻的选择要使振荡频率为 2 MHz 较为适宜，注意不能使振荡频率达到 4 MHz。电阻大小与振荡频率大小之间的关系如图 3-4 所示。同样也要注意电阻应尽量靠近 OSC1 与 OSC2，以改善振荡波形，提高稳定性。在正常情况下，OSC2 引脚上应是方波信号，而 OSC1 引脚上应为三角波信号。该振荡方式 J1 与 J3 系列也适用。

外部振荡方式

在该振荡方式下，外部的振荡源产生振荡信号直接通过 OSC1 引脚输入，引脚 OSC2 悬空。外部振荡信号应是振荡频率稳定的 CMOS 电平振荡信号。

3. $\overline{\text{RESET}}$

为外部复位信号的输入端。当该引脚接收到一逻辑低电平时，会使单片机内部产生复位操作。

4. $\overline{\text{IRQ}}$

为外部中断信号的输入端，低电平有效。当该引脚接收到一下降沿或者低电平时，会使单片机产生外部中断响应操作。

5. PA7～PA0

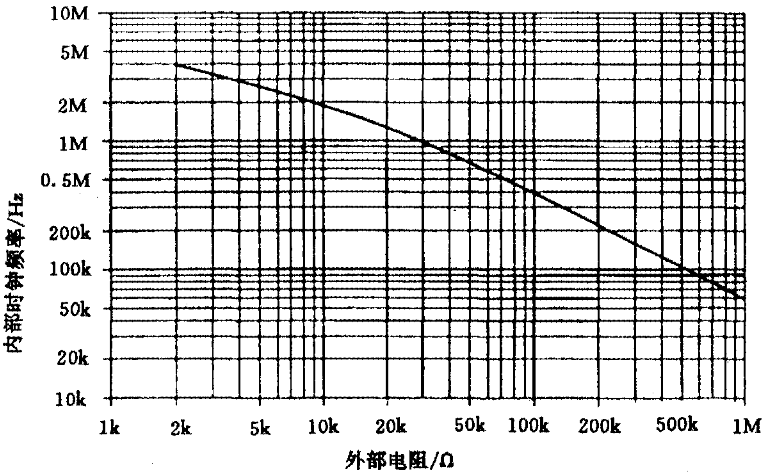


图 3-4　*RC* 振荡频率选择

为 PA 端口的输入/输出引脚。

6. PB5～PB0

为 PB 端口输入/输出引脚。

对于 J3 系列,PB4,PB5 还作为输出比较、输入捕捉功能的输出/输入引脚。

三、MC68HC05J 系列 CPU 结构

MC68HC05J 系列的 CPU 是 MC68HC05 系列标准的中央处理单元,它的内部有累加寄存器 A、变址寄存器 X、栈指针 SP、程序计数器 PC、条件码寄存器 CCR 以及相关的控制电路等,如图3-5所示。

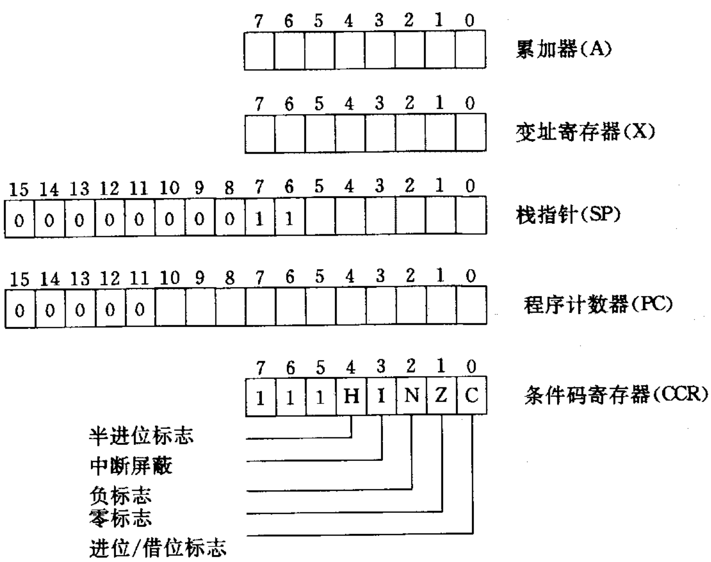


图3-5　MC68HC05J系列CPU结构

累加器 A 同其他 MC68HC05 系列单片机的累加器一样为一个 8 位寄存器，主要功能是存放参加运算的操作数和运算结果。变址寄存器 X 也是一个 8 位寄存器，在变址寻址方式下，该寄存器存入的是地址信息；当然也可用于暂时存放数据。堆栈指针 SP 是 16 位寄存器，用以指示堆栈栈顶的位置。其最大值为 00FFH，最小值为 00C0H。当执行复位操作后，堆栈指针 SP 指向栈底，即 00FFH。每往堆栈压入一个字节的内容，SP 指针加 1；从堆栈内弹出一个字节，SP 指针减 1。当栈指针 SP 指向 00FFH 时，若继续执行出栈操作，则栈指针 SP 会重新置值为 00C0H；同样，如果 SP 已指向堆栈最高处 00C0H 时，若继续执行压栈操作，则栈指针 SP 会重新置为 00FFH。因此，用户要特别注意所设计的程序在执行过程中对堆栈的使用情况，不要出现溢出现象，否则程序将无法正常执行。

程序计数器 PC 用于指示程序执行的位置。对于 J3 系列，它可寻址的上限为 0FFFH，对于 J1，J1A，J2 系列，寻址上限为 7FFH。当执行复位操作后，程序计数器自动装载复位向量。

条件码寄存器 CCR 也是一个 8 位寄存器，其低五位分别存放着 H，I，N，Z，C 标志位。H 是在执行加法操作时，累加器的低四位向高位产生的进位标志，该位为“1”则表示有半进位产生。I 为中断屏蔽位。当 I＝1 时，将屏蔽内部和外部中断，禁止中断产生。该位在单片机复位操作后自动置“1”；在响应某个中断后，也会自动置“1”。N 为负标志，用以反映刚进行的算术操作、逻辑操作或者数据处理与传送的结果是否为负。当结果的最高位 D7 为“1”时，表示结果为负，此时把 N 标志置“1”。Z 为零标志，用以反映刚执行的算术操作、逻辑操作、数据处理以及数据传送的结果是否为“0”。如果结果为“0”，则该标志置“1”。C 为进位或借位标志，用于反映刚执行的算术运算、逻辑操作等是否产生最高进位或借位。如果产生进位或借位，则该标志置“1”。

四、存储器的结构

MC68HC05J 系列的存储器结构与其他 HC05 系列的单片机类似，将 I/O 寄存器、功能部件控制寄存器、数据存储器 RAM 以及程序存储器 ROM 各部分统一编址。将统一的空间分成寄存器部分，RAM 部分，堆栈区，用户 ROM 部分，自检 ROM 部分以及中断向量、复位向量存放部分，如图3-6所示。

图中各部分的起始与终止地址因 J1，J2，J3，J1A 系列而有所不同。详见本章后面几节的说明。

3.1.3 MC68HC05J 系列的并行 I/O 端口

并行端口在 MC68HC05J 系列单片机中共有 14 只引脚，而且都是双向的。每只引脚是相对独立的，既可进行输入，又可进行输出。其中，PA 端口 8 只引脚，PB 端口 6 只引脚。每个端口引脚的内部结构是相同的，都是由一些基本的电路组成，如图 3-7 所示。这些电路包括数据方向寄存器位、数据输出寄存器位、输出缓冲器、输入缓冲器等。

数据方向寄存器 DDR 是一个 8 位的控制寄存器。每个并行端口有一个对应的方向寄存器，用以确定端口的每一通道是输入还是输出通道。J 系列有两个方向寄存器，分别用以确定 PA 端口的 8 只引脚和 PB 端口的 6 只引脚的输入/输出特性。当某位为“1”时，则相应通道为输出方式。

数据输出寄存器用以存放各端口输出的数据。当方向是输出时，该寄存器的内容便直接通过驱动电路输出到引脚；而当方向为输入时，该寄存器的内容不能输出到引脚上。

当要进行数据输入时，方向寄存器首先必须清“0”，输出缓冲器输出高阻抗。当进行读操作

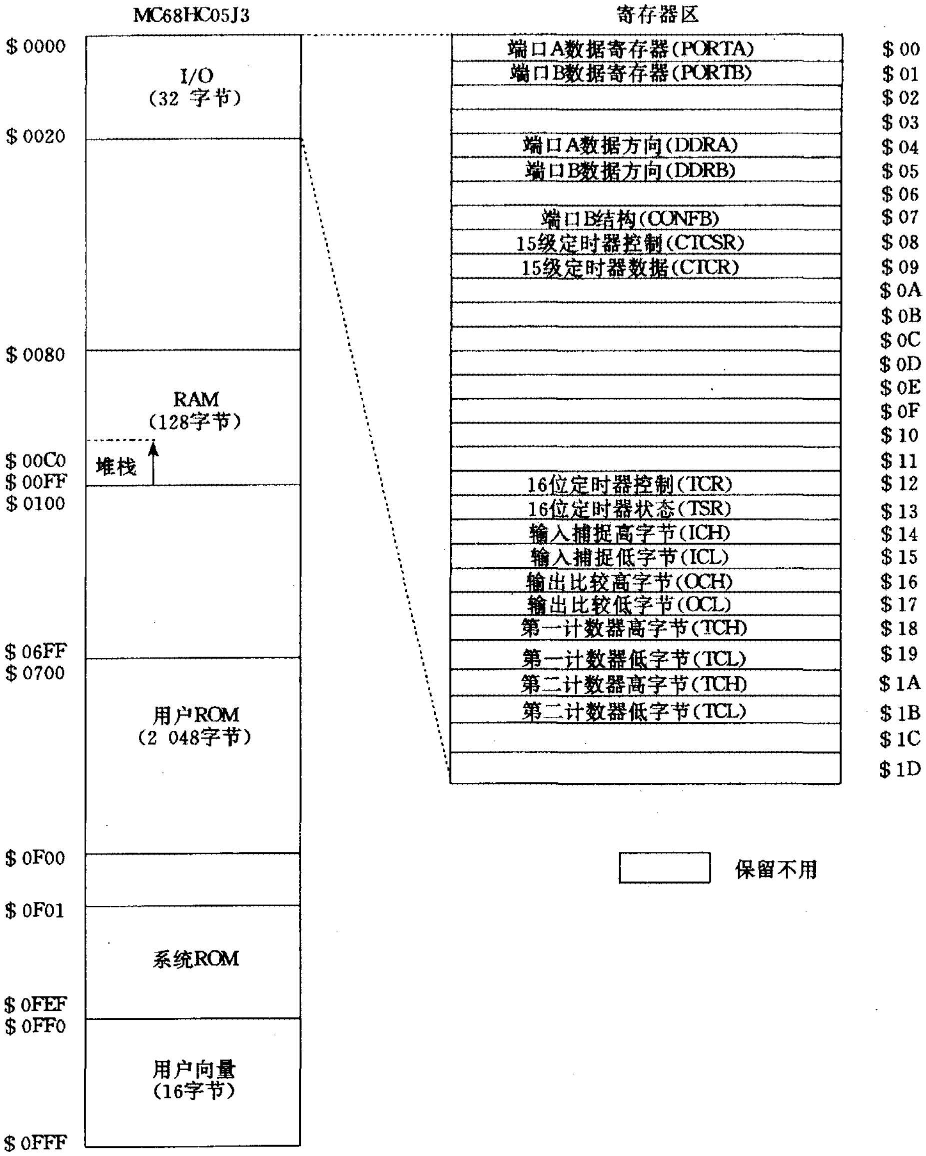


图3-6 J系列存储器分布

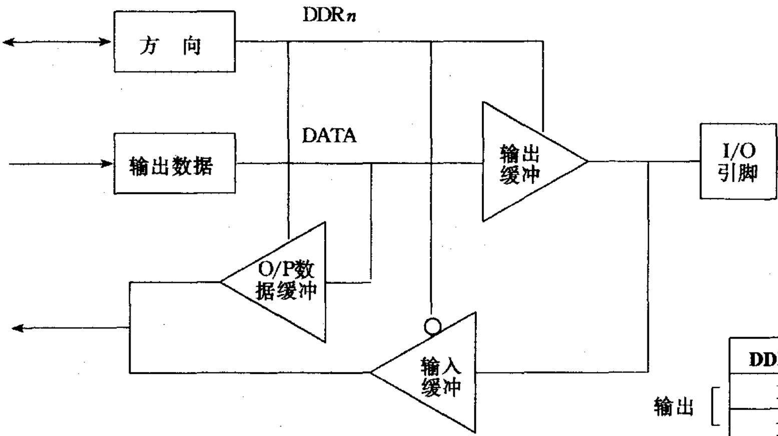
时，输入数据缓冲器则把 I/O 引脚的高低电平数据输入到单片机的内部数据总线上去。

用户在使用 I/O 时，需要注意对每个端口的方向寄存器进行设置以及对数据寄存器的读/写。输出数据时，置方向寄存器的相应位为“1”，然后将数据写入数据寄存器；当要输入引脚电平时，则将方向寄存器相应位清“0”，读数据寄存器即为外部引脚输入的数据。同一时刻某端口各引脚方向寄存器可以不同。

表 3-1 J 系列数据寄存器和方向寄存器地址

寄存器名称	地 址
PA 数据寄存器	00H
PB 数据寄存器	01H
PA 方向寄存器	04H
PB 方向寄存器	05H

对于 J 系列，数据寄存器与方向寄存器的地址分配如表 3-1 所示。



	DDRn	DATA	I/O引脚
输出	1	0	0
	1	1	1
输入	0	0	三态
	0	1	三态

图3-7 J系列I/O结构

J 系列单片机中有的类型除了有以上所述特性外，还具有一些特别的功能。J3 系列的 PB 端口中的 PB5，PB4 还分别作为输入捕捉、输出比较的输入端和输出端，详见本章 MC68HC05J3 单片机章节的叙述。另外，对于 J1A，J3 系列，当有的引脚作为输入时，输入引脚电平的变化可以产生中断。J3 系列的 PB0～PB3，J1A 系列的 PA0～PA3 便具有该功能。

在使用并行端口输出驱动外围电路时，要注意各输出脚的驱动能力。J 系列中驱动能力最强的为 J1A 和 J3 系列。J1A 的 PA 端口(PA7～PA4)和 J3 系列的 PA，PB 端口能够承受 8 mA 灌入电流，而其他的只能承受平均 1.6 mA 的灌入电流以及平均 0.8 mA 的拉出电流。

对于 J1A 系列，当 PA 和 PB 端口处于输入状态时，可选择是否带下拉电阻，这可通过软件设置，也可在掩膜时设定。下拉电阻如选择有效，则当输入引脚悬空时，其输入数据为低电平。这在按键输入电路设计时是很方便的。控制寄存器 PDRA(地址 0AH)和 PDRB(地址 0BH)的各位分别控制着 PA，PB 端口下拉电阻的有效性。

3.1.4 MC68HC05J 系列的复位系统

一、复位的类型

当单片机产生复位操作时，将立即终止任何指令的执行，初始化某些控制单元，之后装载用户定义的复位向量到程序计数器中，重新开始执行用户程序。

复位类型分为内部与外部两种，如图 3-8 所示。外部复位指由单片机的外部引脚 $\overline{RESET}$ 接收到低电平而引起的复位。内部复位单片机自动产生，有三种情况可以激发这种复位操作：监视定时器计数器的溢出，电源由无到有以及非法寻址。

1. 上电复位

当引脚 V_{DD} 检测到一个正跳变时，会自动产生上电复位。在振荡器开始工作后，内部上电复位电路产生 4 064 个时钟周期的延时，并使引脚 $\overline{RESET}$ 在这段时间内保持输出低电平，以便能够复位外围电路。之后将退出复位，开始正常操作。但如果外部引脚 $\overline{RESET}$ 因外围电路而仍然保持低电平，则单片机仍会处于复位状态。

内部上电复位电路只能用于上电复位，不能用于检查电源电压下降，也不能完成掉电复

位。

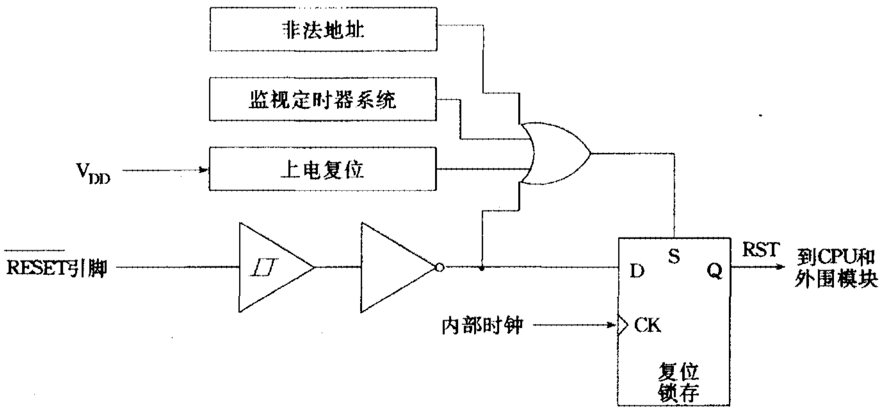


图3-8 复位产生的途径

2. 外部复位

当引脚$\overline{RESET}$接收到多于 1.5 个时钟周期长的低电平信号时，就会激发复位操作。该引脚由一施密特触发器完成信号的触发。高低阀值电压的转化存在滞后，如果$\overline{RESET}$引脚电平低于低阀值电压则产生外部复位，直到该引脚电平高于高阀值电压。

由于 J 系列单片机无低电压复位系统，用户必要时可以利用该引脚配合低电压外围检测电路实现低电压复位。

3. 计算机正常操作监视系统(COP)复位

该系统产生的复位通常称为监视定时器(COP)复位。该系统内部有一个定时器，一旦该定时器溢出就会激发单片机执行复位操作。所以，一旦允许该系统工作后，CPU 就必须及时复位该定时器，以防止其溢出。

通过掩膜时选择以决定是否允许该系统工作。对于 J 系列的 EPROM 芯片，可以通过 MOR 寄存器来选择。MC68HC705J1A 芯片 MOR 寄存器的地址为 7F1H，MC68HC05J1 芯片 MOR 寄存器地址为 700H，MC68HC705J2 和 MC68HC705J3 MOR 寄存器地址为 0F00H。该寄存器处于单片机的 ROM 区。注意，不能使用指令来设置该寄存器，而要用 FCB 伪指令设置。如果要允许监视定时器电路工作，只要置该寄存器的 B0 位为“1”即可。

一旦监视定时器电路开始工作，就不能用软件使之停止，只能用指令清除其定时器，以防止其溢出。为此必须定期使 COPR 寄存器的 B0 位清“0”。对于 J1，J1A 系列，COPR 寄存器地址为 7F0H，而对于 J2，J3 系列，COPR 寄存器地址为 FF0H。该寄存器的地址与 7F0H 或 FF0H 的 ROM 单元地址相同，但并不会造成混乱。因为 ROM 是不可写的，当执行“STA 7F0H”或“STA 0FF0H”指令进行清“0”操作时，是完成对监视定时器电路中的定时器的清“0”；而当进行“LDA 7F0H”或“LDA 0FF0H”读指令时，读入到 A 累加器的是 7F0H 或 0FF0H ROM 单元的内容。

4. 非法地址复位

如果单片机因某种原因，程序执行混乱而跳到 RAM 空间或者跳到不可用的 ROM 空间去取指，则单片机能自动产生复位操作，强迫单片机立即从故障状态跳转出来，重新执行用户程序。

二、复位操作

复位时,单片机便开始进行一系列的初始化工作,包括对 CPU、并行输入/输出端口、定时器以及监视定时器系统。

1. 对 CPU 的初始化

(1) 置堆栈指针为 FFH;

(2) 置状态寄存器的中断屏蔽位为“1”,禁止中断;

(3) 清除低功耗工作状态锁存器,允许 CPU 工作;

(4) 用所定义的复位向量重置程序计数器 PC,复位向量保存在 ROM 0FFEH,0FFFH 单元(J3,J2 系列)或 7FEH,0FFFH(J1,J1A 系列)。

2. 对并行端口的初始化

(1) 把所有端口的方向寄存器清“0”,各端口全部置为输入状态,而数据寄存器保持不变;

(2) 对于 J1A 系列,设置并行端口的下拉电阻有效。

3. 多功能定时器

(1) 清定时器的状态寄存器与控制寄存器;

(2) 清定时器的计数器。

4. 监视定时器系统

清除监视定时器系统的定时器。

3.1.5 MC68HC05J 系列的中断系统

MC68HC05J 系列中断系统的功能很强,这为我们编写控制程序提供了较大的方便。它有软中断、外部中断和定时器中断三种类型。外部中断又包括$\overline{IRQ}$中断和并行输入端口中断。对于 J3 系列,定时器中断还包含输入捕捉中断、输出比较中断及 16 位定时器溢出中断。各种中断中除软中断不可屏蔽外,其他都是可以禁止的。

当某种中断产生后,如响应条件满足,则 CPU 将其寄存器内容保存到堆栈中,中断屏蔽位自动置“1”,以防止对新中断的响应;然后将中断向量取到程序计数器 PC 中,执行中断服务程序,中断服务程序执行完毕,即执行 RTI 指令,将堆栈内容弹出,恢复进入中断前 CPU 寄存器的数据,并继续执行原来的程序段。中断响应的过程如图 3-9 所示。

各种类型中断的中断向量全部存放在地址最高的 ROM 区,如表 3-2 所示。

表 3-2 不同类型中断的中断向量的存放地址

中断类型	中断向量存放地址		优先数	备注
	J1,J1A 系列	J2,J3 系列		
软中断 SWI	7FC,7FD	0FFC,0FFD	最高	
外中断$\overline{IRQ}$,键盘中断	7FA,7FB	0FFA,0FFB		J1A,J3 系列才有键盘中断
15 级定时器中断	7F8,7F9	0FF8,0FF9		
输入捕捉中断	—	0FF6,0FF7		仅 J3 系列有
输出比较中断	—	0FF4,0FF5		仅 J3 系列有
16 位定时器溢出中断	—	0FF2,0FF3	最低	仅 J3 系列有

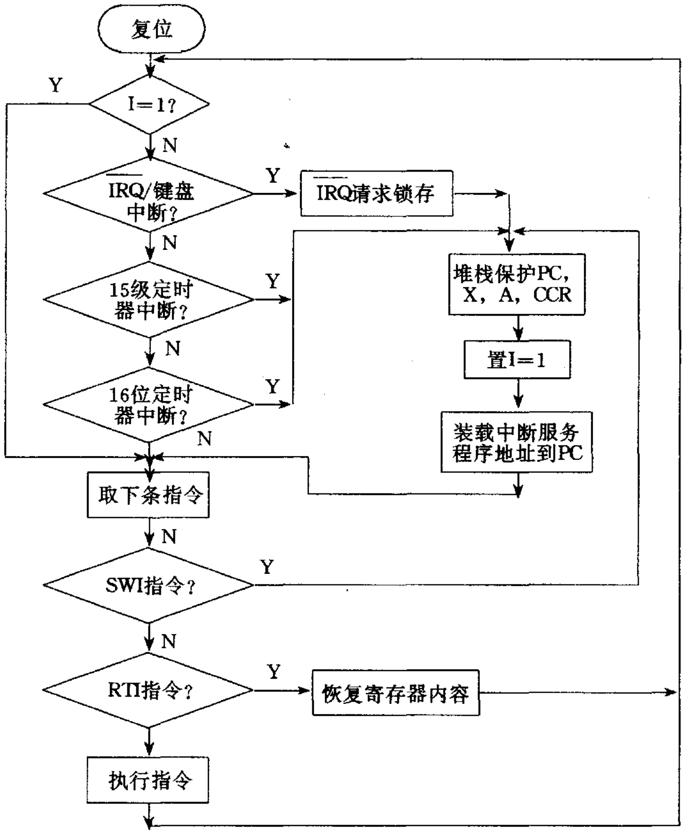


图3-9 中断响应的过程

软中断 SWI 是不可屏蔽的，只要执行指令 SWI 就可进入中断服务程序。该指令在编制开发系统的软件时才常用到，一般用户程序没有必要使用该指令。

外部中断是由外部下降沿信号输入到引脚$\overline{IRQ}$上引起的，J1A 系列的 PA0，PA1，PA2 和 PA3 引脚也可以接收中断信号，J3 系列 PB 端口的 PB0，PB1，PB2，PB3 引脚也具备这样的特性，而其他系列不具备并行端口中断输入功能。由于$\overline{IRQ}$中断与并行端口的中断是共用一个中断向量。因此产生外部中断后，中断服务程序要进行判断是何种中断引起的。

定时器中断的产生详见下节 J 系列定时器的说明，输入捕捉中断、输出比较和溢出中断的产生详见对 J3 系列定时器的说明。

3.1.6 MC68HC05J 系列的多功能定时器

J 系列单片机内部都具有一个 15 位多功能定时器，如图 3-10 所示。使用该定时器可以产生多种定时。该定时器由一个 8 位计数器(地址为 09H)和一个 8 位控制与状态寄存器组成(地址为 08H)。

8 位计数器的计数脉冲是内部时钟 4 分频后形成的，内部时钟由振荡器时钟 2 分频形成，因此，计数脉冲的频率为振荡频率的 1/8。该 8 位计数器的计数值可以随时读出，地址为 09H，但不能用软件改写，只能复位时清“0”。该计数器为加法计数器，每收到一个计数脉冲则加 1。

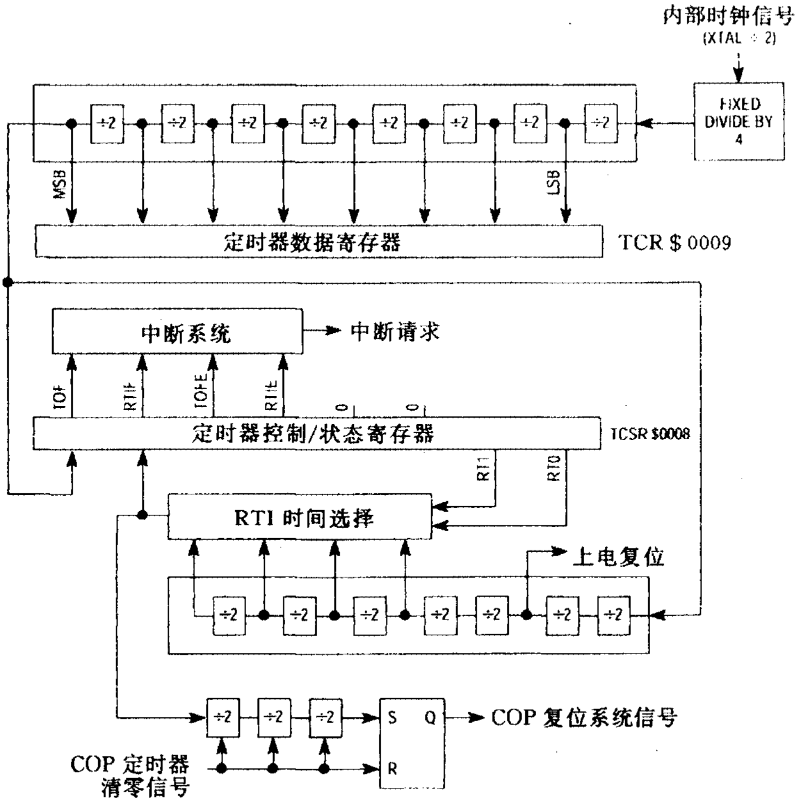


图 3-10　J 系列定时器结构

当计数溢出时，置位溢出标志(在允许的情况下可以引起定时器中断)，该标志可以在定时器的控制与状态寄存器上看到。计数脉冲经过该 8 位计数器(即进行 256 分频)后，又经过进一步的分频，分频数可以是 1/16，1/32，1/64 或 1/128，取何种分频数取决于对定时器的控制与状态寄存器(地址 08H)的设置。每获得一个分频后的脉冲信号，会对控制与状态寄存器(地址 08H)上的标志位置位，并在允许的情况下可以产生中断。该中断称为实时中断，该标志为实时时钟标志。定时器计数信号分频后，所得的脉冲信号又作为监视定时器系统的定时器的计数脉冲，用来确定监视定时器复位清“0”的最小时间。

使用定时器时，最重要的是要注意对其控制与状态寄存器的使用。该寄存器一方面反映定时器工作状态，另一方面又控制定时器工作。该寄存器为 8 位寄存器，每位的定义如图 3-11 所示。

地　址	B7	B6	B5	B4	B3	B2	B1	B0
08H	TOF	RTIF	TOIE	RTIE	TOFR	RTIFR	RT1	RT0

图 3-11　定时器的控制与状态寄存器

各位在复位操作后，除 B1，B0 置“1”外，其他全部清“0”。

TOF——定时器溢出标志

当 8 位计数器的数值由 0FFH 变为 00H 时，该标志位置“1”。要清除该标志，对于 J1A 系列可通过对该寄存器的 B3 位置“1”，或者执行复位操作来完成；对其他系列可直接用软件清“0”该位。

RTIF——实时时钟标志

当溢出时钟信号继续进行 1/16，1/32，1/64 或 1/128 分频之后，也就是说计数脉冲经 12 位、13 位、14 位或 15 位计数溢出后，就会置位该标志位。若要清除该标志位，对于 J1A 系列可通过对该寄存器 B2 位置“1”，或者执行复位操作完成；对其他系列可直接用软件清“0”该位。

TOIE——定时器溢出中断允许

该位控制着溢出时是否允许产生中断。如为“1”，则允许产生溢出中断；否则，禁止溢出中断的产生。

RTIE——实时时钟中断允许

该位控制着是否允许产生实时中断。如为“1”，则允许实时中断的产生。注意，溢出中断与实时时钟中断的中断向量是共用的。在定时器中断服务程序里，要通过 TOF，RTIF 标志位判断是何种中断。

TOFR——溢出标志复位位（仅 J1A 系列有）

当对该位置“1”时，J1A 系列单片机的 TOF 标志自动清“0”。

RTIFR——实时时钟标志清“0”位（仅 J1A 系列有）

当对该位置“1”时，对于 J1A 系列的单片机，可以将实时时钟标志位清“0”。

RT1，TR0——实时时钟时间选择

这两位的数据决定了溢出信号继续分频的系数，也就是确定了实时时钟标志建立的时间，同时也影响了监视定时器溢出的时间。如表 3-3 所示。

表 3-3　RT1，RT0 实时时钟选择

RT1，RT0	RTI 频率	RT1 时间/ms	COP 溢出时间/ms
00	$f_{OP}/2^{14}$	8.2	57.3
01	$f_{OP}/2^{15}$	16.4	114.7
10	$f_{OP}/2^{16}$	32.8	229.4
11	$f_{OP}/2^{17}$	65.5	458.8

注：f_{OP}是内部时钟频率，为振荡频率的 1/2。上面数据是在 $f_{OP}=2$ MHz 情况下计算得到的。

当 CPU 处于低功耗工作状态时，该定时器工作在 STOP 方式下，将停止工作，并清计数器。退出 STOP 方式时，将延时 4 064 个机器周期后恢复工作。在 WAIT 等待方式下，定时器继续工作；如果定时器允许中断，则过一段时间后，将会引起定时器中断，退出 WAIT 等待方式。

3.1.7　MC68HC05J 系列低功耗工作状态

低功耗工作方式是单片机非常重要的特性，尤其是应用在电池供电的仪器、仪表上时，为了延长控制器的工作时间，必须严格控制单片机的功耗，降低控制器的工作电流。单片机在正常工作状态下，其工作电流处于 mA 级；如果大于 5 mA，那么对于电池供电系统，单片机此时

的工作电流就显得太大了。因此，单片机的低功耗工作方式在电池供电系统下显得尤为重要。

MC68HC05J 系列单片机的低功耗工作方式包括 STOP 停止方式、WAIT 等待方式、HALT 等待方式和 Data-Retention 数据保留方式。各系列单片机都具有 STOP 与 WAIT 方式；J2，J3 系列不具有后面两种节电方式；J 系列中仅 J1A 具有 HALT 方式。单片机的工作电流随单片机的工作方式而有所不同。如表 3-4 所示。

表 3-4　不同工作方式时单片机的工作电流

工作状态	J1A 系列	J1 系列	J2 系列	J3 系列
运行	3.0 mA～4.0 mA	2.5 mA～5.0 mA	5.0 mA～7.0 mA	5.0 mA～10 mA
STOP 停止	0.2 μA～10 μA	2.0 μA～30 μA	2.0 μA～30 μA	1 μA～100 μA
WAIT 等待	1.6 mA～2.5 mA	1.2 mA～2.75 mA	1.3 mA～2.5 mA	2.5 mA～4 mA

注：以上数据是在单片机并行 I/O 端口无负载，V_{DD}=5.0 V，室温 25 ℃的情况下测得的。

一、STOP 停止方式

该方式是功耗最低的工作方式。只要执行 STOP 指令即进入该方式。当进入该方式时，单片机的工作状态如下。

(1) 清定时器的控制与状态寄存器，禁止定时器的中断；

(2) 清除任何悬挂着的中断；

(3) 定时器的计数器清“0”；

(4) 中断屏蔽位清“0”，允许外部中断产生；

(5) 停止振荡器的工作；

(6) CPU 与定时器停止工作，监视定时器系统亦停止工作；

(7) 保持其他寄存器数据、输入/输出端口数据以及 RAM 中的数据。

退出该工作方式有两种方法：一是由外部引脚$\overline{\text{RESET}}$输入低电平，引起复位操作而退出；二是由外部引脚$\overline{\text{IRQ}}$输入下降沿信号，引起外部中断的产生而退出。对于 J1A，J3 系列，还可由引脚 PA0～PA3(J1A)或 PB0～PB3(J3)接收中断信号引起外部中断而退出。当因外部复位信号而退出时，单片机要进行复位操作，重新开始执行用户程序；当因外部中断信号而退出时，单片机 CPU 要执行外部中断服务程序。

当单片机退出 STOP 方式时，要经过 4 064 个内部时钟周期的延时后，才开始正常工作。

二、WAIT 等待方式

要进入该方式，只要执行指令 WAIT 即可。当进入该方式后，单片机的工作状态如下。

(1) 清中断屏蔽位，允许中断产生；

(2) 允许外部中断；

(3) 停止 CPU 的时钟，CPU 停止工作；

(4) 内部振荡器仍然工作；

(5) 定时器继续工作；

(6) 保持其他寄存器的数据、并行 I/O 端口数据以及 RAM 中的数据。

退出 WAIT 等待方式可以通过内部中断、外部中断或复位三种方法实现。其中，内部中断包括定时器中断以及输入捕捉、输出比较、定时器溢出中断等多种；外部中断包括$\overline{\text{IRQ}}$中断

和并行端口中断;复位包括外部$\overline{RESET}$复位以及内部监视定时器溢出复位。退出后,CPU 自动取对应的中断向量或复位向量,执行相应的程序。

三、HALT 等待方式

对于 J1A 系列单片机,可以掩膜选择禁止 STOP 方式。如果在禁止 STOP 方式下,程序中有 STOP 指令,则执行到该指令后单片机不会进入 STOP 方式,而是进入 HALT 等待方式。HALT 方式与 WAIT 等待方式实质上是相同的。这两种工作方式下单片机的工作状态是相同的;不同的只是在退出时,HALT 方式同 STOP 方式一样,需要 4 064 个内部时钟周期的延时才能恢复正常运行。

这三种低功耗方式的工作流程图如图 3-12 所示。

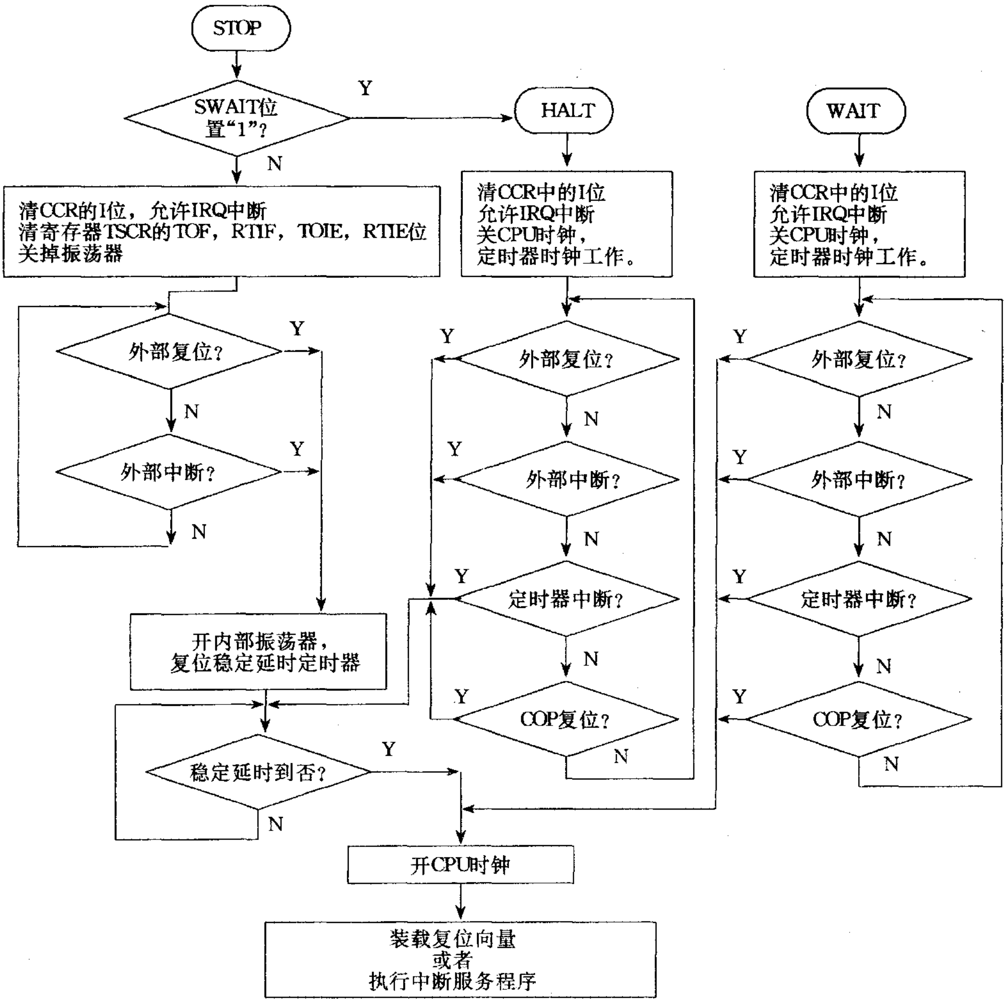


图3-12 STOP,WAIT,HALT流程图

四、Data-Rentention 数据保留方式

在数据保留方式时,单片机已全部不能工作,仅仅只能保持当前 RAM 与 CPU 寄存器的数据。该方式只能通过外部条件强行进入。当复位引脚$\overline{RESET}$被强行拉低为低电平、V_{DD}拉低(最低可达 2.0 V)时,单片机便进入该工作方式。注意,$\overline{RESET}$在该方式时要保持低电平。

当要退出该方式时，必须恢复 V_{DD}的电压值，使引脚$\overline{RESET}$电平升高为高电平，之后单片机执行复位操作，重新开始执行用户程序。

3.2 MC68HC05J1A 单片机特点

MC68HC05J1A 是 MOTOROLA 公司最新的 J 系列单片机产品。在 J 系列的各种单片机中，J1A 系列单片机的性能价格比最高，是近年来应用得最多的一种单片机。该系列单片机除了具有上节讲述的 J 系列单片机的功能外，还有其独特之处。本章节将介绍一下其独特之处。

MC68HC05J1A 主要特性如下。

(1) 1 240 字节 ROM，64 节字 RAM；

(2) 堆栈容量为 64 字节，堆栈区与 RAM 区重叠；

(3) 14 个双向输入/输出端口，各 I/O 引脚可软件设置下拉电阻；PA 端口具有 8 mA 灌入电流的承受能力；有 4 个 I/O 引脚可以产生中断信号；

(4) 可软件设置外部中断$\overline{IRQ}$的允许与禁止；

(5) 无最小时钟频率选择；

(6) STOP，WAIT，HALT，Data-Retention 多种低功耗工作方式；

(7) 可掩膜选择振荡方式以及晶振并行电阻，STOP 使用，外部中断激活方式，COP 系统的使用，并行端口下拉电阻的使用。

3.2.1 MC68HC05J1A 存储器结构

J1A 系列的存储器也分为寄存器部分，数据存储器 RAM 部分，用户程序存储器 ROM 部分，自检程序 ROM 部分以及中断复位向量 ROM 部分。如图 3-13 所示。

存储空间的 00H～1FH 共 32 字节为各个寄存器，其中包括并行端口、定时器、外部中断控制等控制与状态寄存器。

存储空间从 0C0H～0FFH 共 64 字节的地方为数据存储区以及堆栈区。这里要特别注意，J1A 的堆栈区与 RAM 重叠使用同一空间，所以，自定义变量不能太多，不然堆栈区将不够用；另外，变量的定义从 C0H 开始顺序向下，因为堆栈的栈底为 FFH。在使用堆栈时，要防止堆栈顶触及变量定义的 RAM 区。

300H～7CFH 的存储区共 1 232 字节为用户程序存储区。

7D0H～7EFH 为单片机自检程序用空间。

7F0H～7FFH 为中断向量与复位向量存放空间。在该空间中，用户只能用 7F8H～7FFH 的向量；另外，7F0H 单元会对监视定时器清“0”。读该单元得到的是该 ROM 中的数据。

3.2.2 MC68HC05J1A 的并行 I/O 端口

J1A 系列的 I/O 端口特性大部分与其他 J 系列单片机的相同。其结构如图 3-14 所示。

它的主要特点是驱动能力强，下拉电阻可选择使用，并行输入端口可产生中断。

PA 端口的 PA7～PA4 四个引脚可以承受 8 mA 灌入电流，在该负载下，其输出低电平的电位仍可小于 0.4 V。另外，PA 端口的 PA3～PA0 可以产生外部中断，特性与$\overline{IRQ}$中断一样，中断向量也与$\overline{IRQ}$中断一样（为 7FAH，7FBH）。该中断功能可以通过掩膜选择。对于

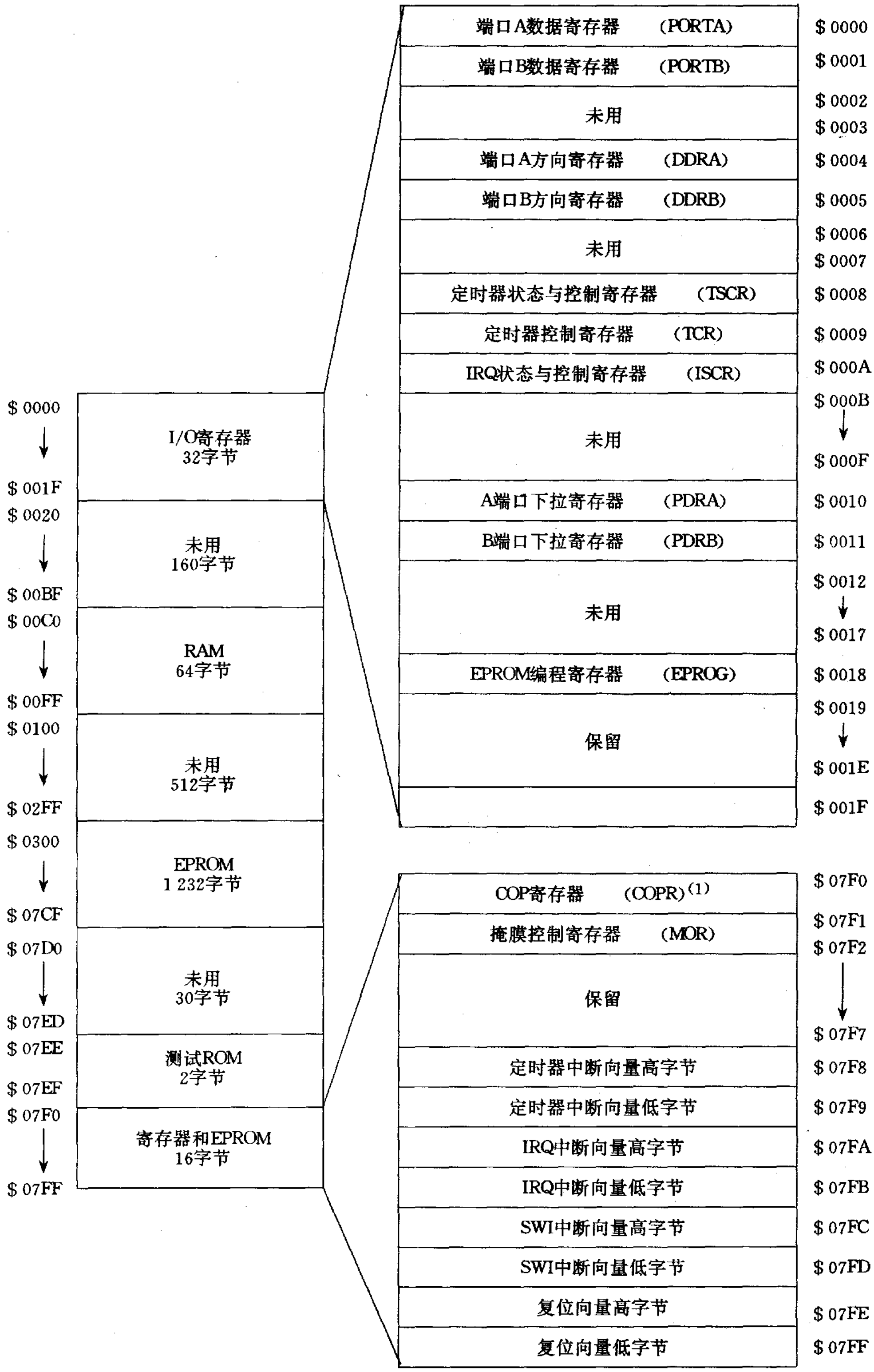


图3-13　J1A系列存储器的结构

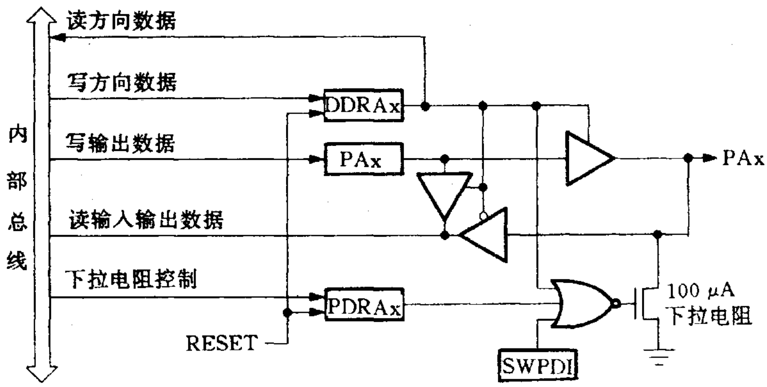


图 3-14　J1A 系列 I/O 端口结构

68HC05J1A EPROM 系列，可以通过掩膜选择寄存器 MOR(地址 700H)来实现设置。该中断功能常用来实现控制器按键矩阵的读入，所以也称这种中断为按键中断。该中断为高电平有效。

J1A 的 PA，PB 两端口的使用除了数据寄存器与方向寄存器外，还要注意设置下拉电阻的控制寄存器。对于 PA 端口，其地址为 10H；对于 PB 端口，其地址为 11H。这两个寄存器的各位分别决定相应端口各引脚的下拉电阻是否有效，置“1”表示下拉电阻无效；为“0”表示下拉电阻有效。由图 3-13 看出，当 I/O 引脚处于输出状态时，下拉电阻是无效的。另外，在掩膜时或在 MOR(地址 700H)寄存器中可以选择下拉电阻。一旦掩膜禁止，那么，无论下拉电阻控制寄存器的数据如何，下拉电阻都是无效的。

3.2.3　MC68HC05J1A 的中断系统

J1A 系列单片机的中断方式包括软中断 SWI，外部中断 $\overline{IRQ}$，PA3～PA0 按键中断，15 级多功能定时器的溢出中断与实时时钟中断。中断向量包括软中断、外部中断、定时器中断三种。其外部中断因具有 PA3～PA0 按键中断而同其他系列有所不同。

其外部中断结构如图 3-15 所示。

PA3～PA0 产生的中断可以由掩膜选择禁止与允许。注意，PA3～PA0 的中断与 $\overline{IRQ}$ 引脚中断不同。$\overline{IRQ}$ 是低电平有效，当 $\overline{IRQ}$ 引脚接收到负跳变信号或低电平信号时，将会激发中断的产生；而 PA3～PA0 中断是高电平有效，当 PA3～PA0 引脚中的一个接收到边缘正跳变信号或高电平信号时，就会激发中断产生。两种中断的激发方式都统一由掩膜时选择或者通过 MOR 寄存器选择，激发方式可以是电平触发或边沿触发，也可以仅仅是边沿触发。如果允许电平触发，则必须注意在外部中断服务程序退出前，使其引脚的电平恢复到相应电平，对 $\overline{IRQ}$ 要恢复到高电平，对 PA3～PA0 要恢复到低电平；否则，CPU 又会再次进入外部中断服务程序。

J1A 单片机的中断同其他 J 系列单片机相比，还多一个外部中断控制与状态寄存器，其地址为 0AH。如图 3-16 所示。

IRQE——外部中断允许。为“1”则允许；为“0”则禁止外部中断产生，包括 $\overline{IRQ}$ 与 PA3～PA0 中断。

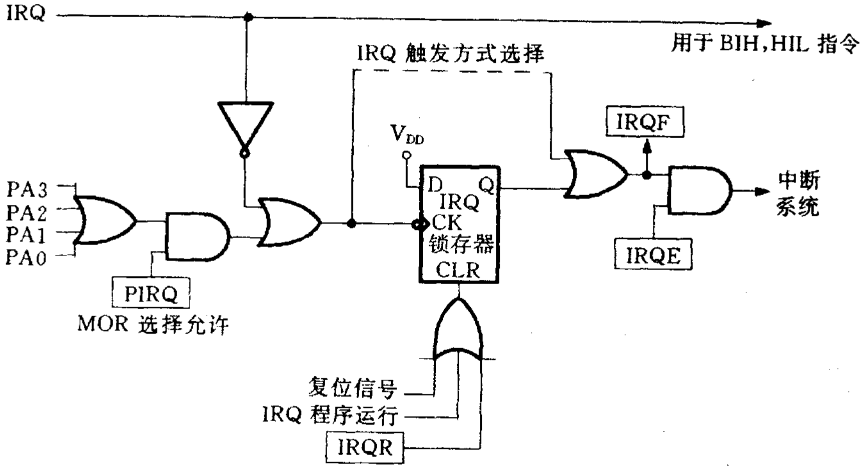


图 3-15　J1A 系列外部中断结构

IRQF——外部中断产生标志。当该位为"1"时表示外部中断已产生;当复位或该中断得到响应,或者对该寄存器的 B1 位置"1"时,可清除该标志。

IRQR——外部中断标志清零位。当该位置"1"时,可清除 IRQF 标志。

地　址	B7	B6	B5	B4	B3	B2	B1	B0
0AH	IRQE	0	0	0	IRQF	0	IRQR	0

图 3-16　外部中断控制与状态寄存器

3.2.4　MC68HC05J1A 的掩膜选择

前面讲述的有关功能中,许多都可以通过掩膜进行选择。对于 J1A 系列单片机来说,这些可选择的功能包括以下几种。

1. 振荡方式

可以选择采用晶振(陶瓷)振荡方式或是 *RC* 阻容振荡方式(包括外部振荡方式)。

2. 晶振电路 OSC1,OSC2 并联电阻的设置

在 J 系列单片机的振荡电路中,在引脚 OSC1 与 OSC2 之间经常要并联 1 个 10 MΩ 的电阻。为了方便,可以掩膜选择使用内部电阻,不需在外部连接。

3. STOP 停止方式的允许与禁止

当选择禁止后,就算程序执行 STOP 指令,也只能使单片机进入 HALT 方式,或者说是进入 WAIT 等待方式。

4. 外部中断激活方式

可以选择仅边沿触发,也可以选择边沿或电平触发。

5. PA,PB 端口下拉电阻的设置

这些 I/O 引脚具有下拉电阻,下拉电阻是否有效则可掩膜选择。

6. 监视定时器系统的选择

用户可以根据需要掩膜决定是否使用监视定时器系统。

7. PA 端口中断功能的选择

对 PA 端口的 PA3～PA0，通过掩膜选择决定是否能激发外部中断。

对于 EPROM 系列，掩膜选择是通过 MOR 寄存器(700H)来实现的。

3.2.5 MC68HC705J1A 特性

在开发 J1A 单片机系统时，需要使用 EPROM 型的 68HC705J1A 芯片。因为 MC68HC05J1A 芯片的程序存储空间为 ROM，在制造芯片时程序就已被写入到单片机内部 ROM 了，不可再改写。因此，我们在开发时就不能使用 68HC05J1A 芯片，而只能用 68HC705J1A 芯片。该芯片的程序存储空间是 EPROM，可以改写。这种芯片又分成一次性编程(OTP)型和多次编程型两种芯片。OTP 芯片只可写一次程序，不可多次改写。

一、68HC705J1A 掩膜寄存器

68HC705J1A 芯片的掩膜选择是通过 MOR 寄存器实现的。MOR 寄存器地址是 7F1H，为 EPROM 空间。图 3-17 是 68HC705J1A 的掩膜寄存器各位的意义。

地 址	B7	B6	B5	B4	B3	B2	B1	B0
7F1H	SOSCD	EPMSEC	OSCRES	SWAIT	SWPDI	PIRQ	LEVEL	COPEN

图 3-17 68HC705J1A 的掩膜寄存器

SOSCD——选择上电复位或从 STOP 中复位的延时时间。该延时时间是为了确保振荡器稳定工作后才进行操作。该位为“1”表示延时时间为 16 个 CPU 时钟周期；为“0”则表示延时时间为 4 064 个 CPU 时钟周期。

EPMSEC——片内 EPROM 数据保护位。当该位为“1”时，表示片内 EPROM 数据不可由外围电路读出，以保证片内程序不被人抄录；当该位为“0”时，该保护不起作用。

OSCRES——选择晶振方式下 OSC1 与 OSC2 之间片内 2 MΩ 电阻是否有效。该位为“1”则有效。

SWAIT——STOP 工作方式禁止位。该位为“1”表示无 STOP 功能，此时如执行 STOP 指令也只能使单片机进入 HALT 方式。

SWPDI——I/O 端口片内下拉电阻禁止位。该位为“1”表示禁止使用片内下拉电阻。

PIRQ——PA 端口的 PA3～PA0 中断功能允许。该位为“1”表示允许中断。

LEVEL——外部中断激活方式选择。该位为“1”表示外部中断可以为低电平或下降沿触发；该位为“0”时，表示仅能由下降沿触发。

COPEN——COP 监视定时器功能允许。该位为“1”时允许。

二、68HC705J1A 的片内 EPROM 编程

68HC705J1A 的程序空间为 EPROM。300H～7CFH 的程序空间和 7F8H～7FFH 中断复位向量区都是可编程的。使用开发器或者专用编程器可以将用户程序写入到其片内 EPROM 中。无特殊情况时，用户不需自己编程来灌写程序。但用户可以使用自己的软件对相应 EPROM 单元进行编程操作，用以存放有关数据。当然，一旦对某单元编过一次程，就不能再进行修改了。对于非 OTP 型 EPROM，可用紫外光擦除后再次编程。当然，此时还要重新灌写你

的程序。

对 EPROM 的编程操作(包括 MOR 寄存器的编程)是通过 EPROG 寄存器来实现的。其地址为 18H,定义如图 3-18 所示。

地 址	B7	B6	B5	B4	B3	B2	B1	B0
18H	—	—	—	—	—	ELAT	MPGM	EPGM

图 3-18 EPROG 寄存器的位定义

ELAT——EPROM 地址锁存位。一旦该位被置"1",则单片机将为编程操作自动锁存位编程的地址与数据信息。要编程时,该位必须置"1"。编程完毕,将该位清"0",并退出编程。

EPGM——EPROM 编程启动开关。当该位置"1"时,会将编程电压接通,开始执行编程。该位的置位只有在 ELAT 已经置"1"的前提下进行。清除该位则停止编程。ELAT 位的清"0"操作会自动完成对该位清"0"。

MPGM——掩膜寄存器 MOR 编程控制。该位置"1"表示对 MOR 寄存器编程。

编写一字节 EPROM 的步骤如下:

(1) 改变外围电路,将 16.5 V 编程电压加到$\overline{IRQ}$引脚上;

(2) 将 ELAT 位置"1";

(3) 执行 STA 写操作指令,送数据到对应 EPROM 单元;

(4) 将 EPGM 位置"1";

(5) 延时 4 ms;

(6) 清除 ELAT 和 EPGM 位,即完成编程一字节操作。

对 MOR 寄存器的编程步骤为:

(1) 将编程电压加到$\overline{IRQ}$引脚上;

(2) 将 ELAT 位置"1";

(3) 对 MOR 寄存器执行写操作;

(4) 将 MPGM 位置"1";

(5) 延时 4 ms;

(6) 清除 ELAT 和 MPGM 位。

3.3 MC68HC05J1 单片机

MC68HC05J1 单片机是 J 系列中功能最简单的,但它同样具有 J 系列单片机的主要部件,只是它的存储空间没有 J 系列的其他单片机大,只有 1 040 字节的用户 ROM。它的主要特点是:

(1) 64 字节片内 RAM,64 字节的堆栈;数据 RAM 区与堆栈区共用同一存储空间;

(2) 1 040 字节程序存储器 ROM;

(3) 14 个双向 I/O 并行端口;

(4) 15 级多功能定时器;

(5) STOP,WAIT,Data-Retention 低功耗工作方式;

(6) 监视定时器系统；

(7) 多种振荡方式，包括晶振、陶瓷振荡，*RC* 阻容振荡，外部振荡；

(8) 可掩膜选择振荡方式，外部中断激活方式，STOP 指令的使用，COP 监视定时器系统的使用。

3.3.1 MC68HC05J1 存储器结构

MC68HC05J1 单片机的存储器同样分为控制寄存器区，数据存储 RAM 区（包含堆栈区），用户程序存储 ROM 区，自检程序存储区和中断、复位向量存储区。其存储结构如图3-19所示。

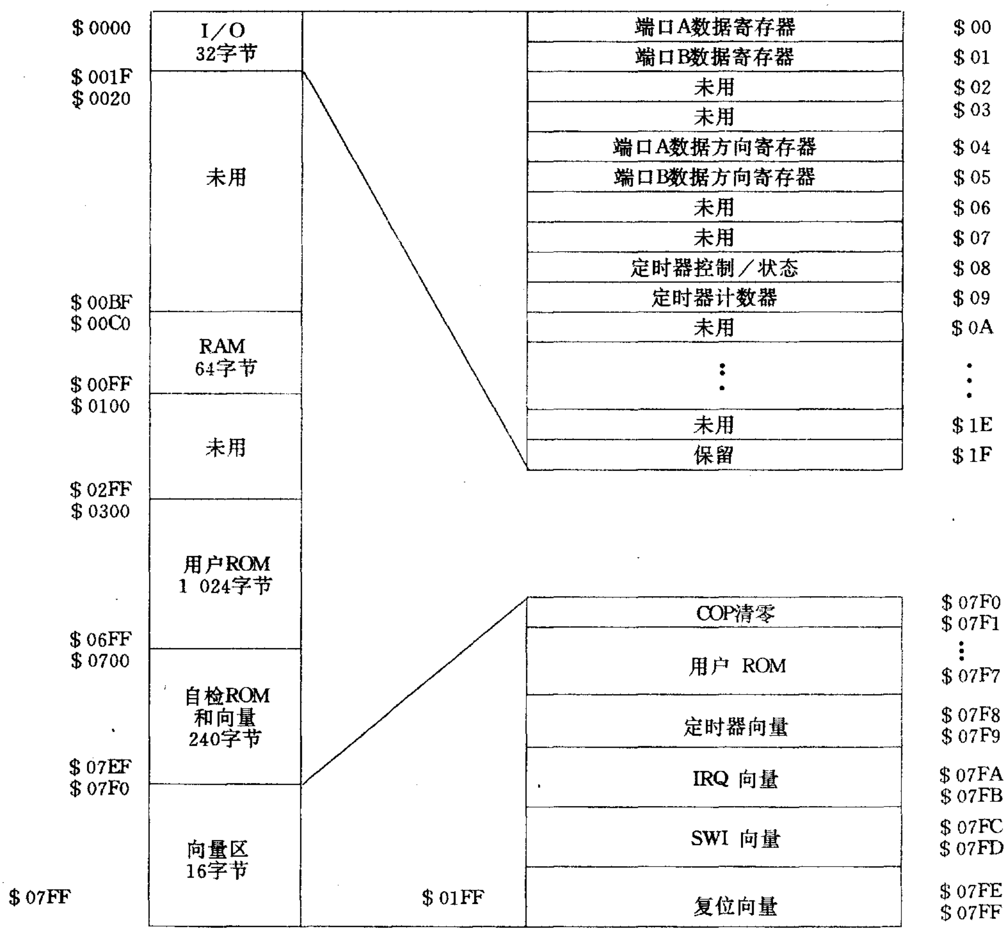


图3-19 MC68HC051J1存储器结构

00H～1FH 为控制寄存器区，包括并行端口的数据寄存器与方向寄存器、定时器控制与状态寄存器等。

0C0H～0FFH 为数据存储 RAM 区，共 64 字节。同 J1A 系列相同，其堆栈区亦是占用该区，所以要注意使用 RAM 单元的数量以及堆栈的深度。

300H～6FFH 为用户程序存储 ROM 区，共 1 024 字节。

700H～7EFH 为自检程序存储 ROM 区。

7F0H～7FFH 为中断与复位向量的存储区。其中，7F0H 单元与 J1A 类似，对 7F0 地址单元进行清"0"操作能使监视定时器复位；读该单元得到的是该 ROM 单元中存储的数据。

3.3.2 MC68HC05J1 单片机的仿真

在开发 MC68HC05J1 单片机的控制器时，必须要使用 EPROM 型的单片机，而 J1 无 EPROM，因此采用 J2 的 68HC705J2 来进行仿真。J1 与 J2 的内存分布是不相同的，但用户不必修改程序以适应 J2 的内存分布，只需在将 J1 系列的程序写到 68HC705J2 中时，注意设置 68HC705J2 的 MOR 寄存器，以使 J2 处于 J1 模式下即可。详见下节对 MC68HC05J2 系列单片机的介绍。同样，J1 的掩膜选择也是由 MOR 寄存器实现的。不过要注意，用 68HC705J2 去仿真 J1 时，振荡方式无 *RC* 振荡选择；再有，STOP 禁止功能也不能选择，STOP 必须是有效的。

3.4 MC68HC05J2 单片机

MC68HC05J2 是在 J1 的基础上，扩大了 J1 的用户存储空间，不但用户 ROM 扩大一倍，RAM 也差不多扩大了一倍。另外，用 MC68HC705J2 单片机可以仿真 J1 单片机。J2 单片机特点如下：

(1) 112 字节 RAM，64 字节深度堆栈；数据 RAM 区与堆栈区共用同一存储区；

(2) 2 048 字节程序存储器 ROM；

(3) 14 个双向 I/O 并行端口；

(4) 15 级多功能定时器；

(5) 低功耗工作方式：STOP，WAIT，Data-Retention；

(6) 监视定时器系统；

(7) 多种振荡方式：晶振、陶瓷振荡、外部振荡、无 *RC* 阻容振荡方式；

(8) 可掩膜选择外部中断激活方式以及监视定时器系统有效。

3.4.1 MC68HC05J2 存储器结构

MC68HC05J2 系列的存储器结构如图 3-20 所示。其程序存储器空间可达 2 KB。

00H～1FH 单元为控制寄存器区。

90H～FFH 单元为 RAM 数据存储区及 64 字节的堆栈区，共 112 字节。

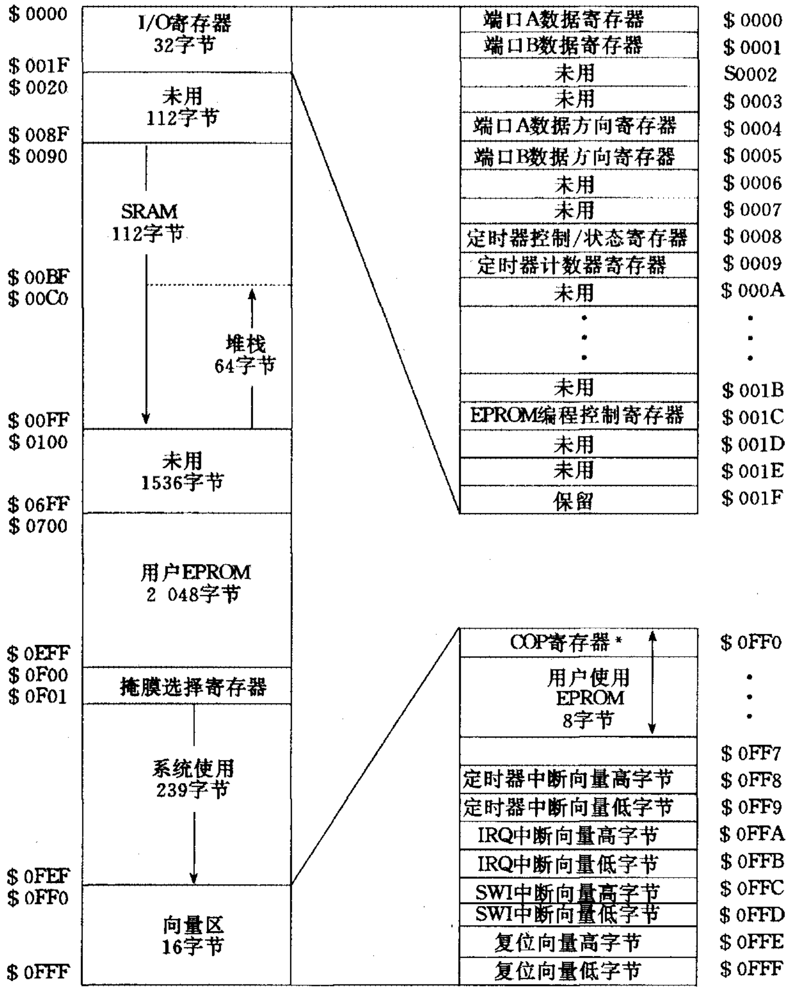
700H～0EFFH 单元为用户程序存储区。

0F00H～0FEFH 单元为自检程序存储区；对于 68HC705J2 系列为自装载程序存储区。

0FF0H～0FFFH 单元为中断向量、复位向量存储区。其中，0FF0H 单元也用于监视定时器清"0"。J2 系列未用 0FF0H～0FF7H 空间，用户可以在该空间内存放只读数据。

3.4.2 MC68HC705J2 单片机

在实际开发时，我们需要使用 EPROM 型单片机来仿真。MC68HC705J2 的程序存储空间为 EPROM 型。



* 向地址0FF0H的最低位(0位)写“0”可清除监视定时器;读0FF0H返回用户EPROM中的数据。

图3-20　MC68HC05J2存储器结构

一、片内 EPROM 的编程

使用开发器或者专用编程器可以将用户程序写入到 MC68HC705J2 芯片中。用户也可以自己根据技术资料做一个简单的 J2 编程器。详见第四章的说明。

用户亦可以自己对未编程的 EPROM 单元进行编程操作,这在存储生产序号或少量其他数据时常常会用到。当然,一旦写上数据就不可再用程序更改了。

对 EPROM 的编程操作可以通过控制寄存器 PROG 来实现,其地址为 1CH。该寄存器位定义如图 3-21 所示。

地址	B7	B6	B5	B4	B3	B2	B1	B0
1CH	0	0	0	0	0	LATCH	0	EPGM

图 3-21　控制寄存器 PROG 的位定义

LATCH——EPROM 地址锁存位。该位可读可写。一旦该位置“1”,则单片机为编程自动锁存住编程的地址与数据信息。要进行编程必须使该位置“1”,编程完毕后要将该位清“0”。

EPGM——EPROM 编程开始开关。该位置“1”时,将编程电压接通,开始进行编程。清除

该位则停止编程。LATCH 清“0”操作会自动完成该位的清“0”。

下面是编写一个字节 EPROM 的步骤：

(1) 改变外围电路，使 16.5 V 编程电压加到 $\overline{IRQ}$ 引脚上；

(2) 将 LATCH 位置“1”；

(3) 执行 STA 写操作指令，送数据到对应的 EPROM 单元中去；

(4) 将 EPGM 位置“1”，开始编程该单元；

(5) 延时 4 ms；

(6) 清 LATCH 位，即完成一个字节的编程。

二、MC68HC705J2 掩膜寄存器 MOR

对于 EPROM 系列，其掩膜选择是通过掩膜寄存器(MOR)来实现的。J2 掩膜寄存器位定义如下。

地址	B7	B6	B5	B4	B3	B2	B1	B0
F00H*	—	—	—	—	—	J1	IRQ	COP

* 对于仿真 J1 模式，地址为 700H。

图 3-22 MC68HC705J2 的掩膜寄存器 MOR

J1——MC68HC05J1 系列仿真模式选择。由于 J1 系列无 EPROM，J2 系列单片机就可以通过将该位置“1”，使之变成 J1 系列单片机来使用，无需修改程序。一旦该位置“1”，其存储器的结构就变为 J1 系列的结构。

注意，当使用 J2 来仿真 J1 工作时，其振荡方式将没有 *RC* 阻容振荡方式选择以及 STOP 功能禁止选择。另外，MOR 的地址也变为 700H。

IRQ——外部中断激发方式选择。该位为“1”，下降沿或低电平信号都可激发外部中断；该位清“0”时，$\overline{IRQ}$ 仅能使用下降沿信号激发中断。

COP——监视定时器系统的使用。该位置“1”，则允许监视定时器系统的使用；否则就禁止。

3.5 MC68HC05J3 单片机

MC68HC05J3 系列单片机是 J 系列中功能最强的，它不但具有 J 系列的 15 级多功能定时器，还具有一个功能强大的 16 位定时器，该定时器在其他 68HC05C 系列等高性能单片机上才具有。通过该定时器可以实现输出比较、输入捕捉等功能，能较方便地编写测频、调频信号输出功能的程序。J3 系列单片机的特性如下：

(1) 2 048 字节的用户程序存储器 ROM；

(2) 128 字节的数据存储器 RAM，64 字节堆栈；堆栈区与数据 RAM 区共享；

(3) 14 只双向输入/输出脚，每只 I/O 引脚驱动能力强，具有 8 mA 灌入电流的承受能力；

(4) 具有键盘中断功能，PB 端口能激发外部中断；

(5) 具有双定时器：15 级多功能定时器 1 和 16 位多功能定时器 2；

(6) 具有输入捕捉、输出比较功能；

(7) 低功耗工作方式：STOP 停止方式与 WAIT 等待方式；

(8) 监视定时器系统；

(9) 多种振荡方式：晶体、陶瓷振荡、*RC* 阻容振荡、外部振荡；

(10) 可掩膜选择振荡方式、STOP 方式禁止功能、$\overline{IRQ}$中断激活方式以及监视定时器系统。

3.5.1 MC68HC05J3 存储器结构

J3 系列单片机的可用存储器空间在 J 系列中是最大的，其程序存储器空间与 J2 系列相同，但其 RAM 空间可达 128 字节。J3 系列存储器分布图如图 3-23 所示。

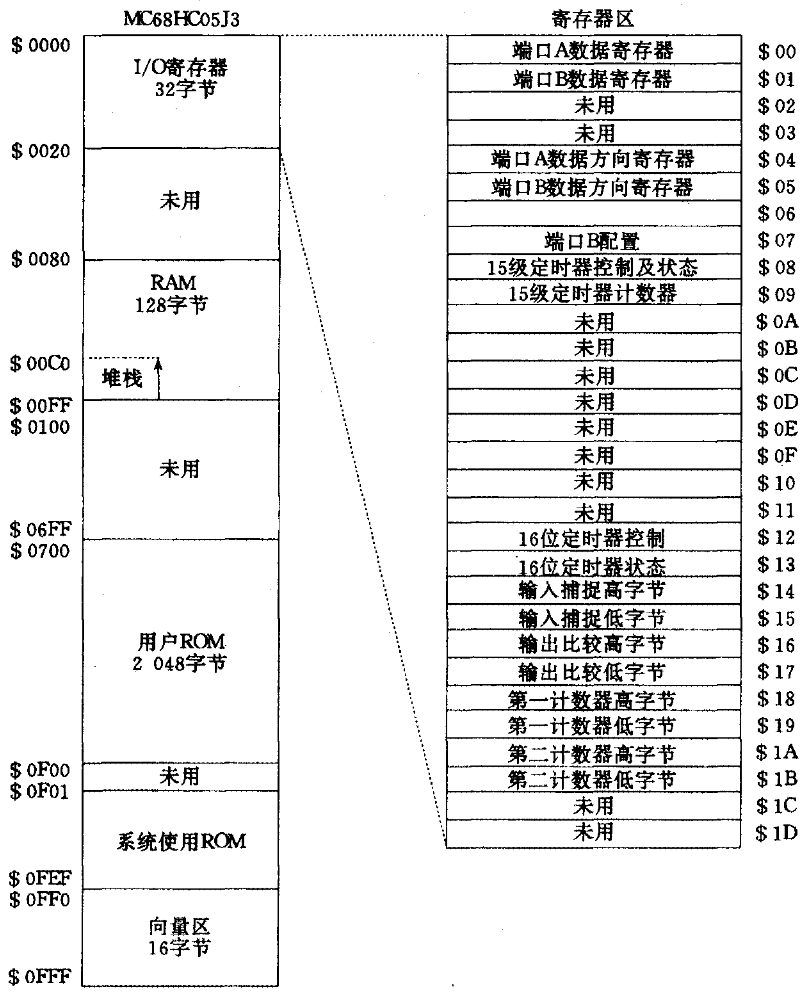


图3-23 J3系列存储器分布

00H～1FH 为控制寄存器区。

80H～FFH 为数据存储区 RAM，共 128 个字节，其中包括 64 字节的堆栈区。

700H～0EFFH 为用户程序存储区，共 2 048 字节。

0F00H～0FEFH 为自检程序存储空间。

0FF0H～0FFFH 为中断与复位向量的存储区。

3.5.2 MC68HC05J3 的 I/O 端口

J3 系列的并行输入/输出端口功能在 J 系列中是最强的。其驱动能力强，能够承受 8 mA 灌入电流。PB 端口的 PB0～PB3 四个引脚具有外部中断激发功能，而 PB4，PB5 还可分别作为其内部 16 位定时器的输入捕捉功能的输入脚和输出比较功能的输出脚。关于输入捕捉与输出比较功能，详见下一节。

一、PB 端口的中断功能

J3 系列的 PB 端口功能较强，其中，PB0～PB3 具有中断功能，如图 3-24 所示。

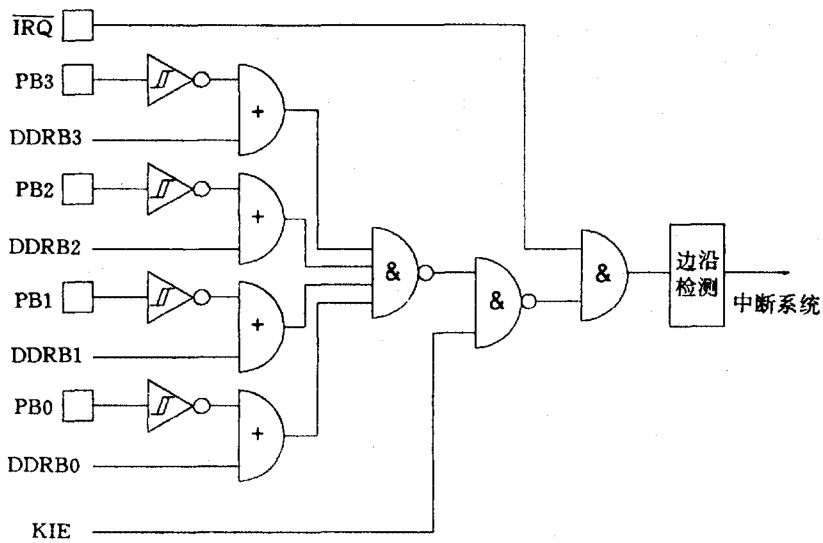


图 3-24 J3 系列 PB 端口中断功能

当 PB0～PB3 作为输入端口时，可以软件控制其是否能用外部信号上升沿激发外部中断。该中断与$\overline{IRQ}$中断是用同一中断向量，因此在外部中断服务程序中，应首先判断是何种信号激发的。另外，PB0～PB3 引起的中断还可以使系统从低功耗工作方式退出。

由于 PB 端口具有多种功能，控制其功能的控制寄存器除了有数据寄存器和方向寄存器外，还具有另一控制寄存器 CONFB，用来控制 PB 端口的多功能，其地址为 07H，各位意义表示如图 3-25 所示。

地址	B7	B6	B5	B4	B3	B2	B1	B0
07H	KSF	KIE	TCAP	TCMP	0	0	0	0

* 复位后数值为 0。

图 3-25 控制寄存器 CONFB

KSF——PB0～PB3 激发中断标志。为“1”表示中断产生；为“0”则表示该中断未产生。该标志常用作区别$\overline{IRQ}$中断与 PB 端口中断。

KIE——PB 端口 PB0～PB3 中断功能的控制位。为“1”表示该功能允许；否则就禁止该功能。

TCAP——PB5 引脚输入捕捉功能选择。如果该位为“1”，则会强制使 PB5 引脚作为输入引脚，而不管 PB5 的方向寄存器如何设置，并且具有输入捕捉功能；该位为“0”则表示 PB5 作为普通双向 I/O 使用。

TCMP——PB4 引脚输出比较功能选择。如果该位为“1”，则会强制使 PB4 处于输出状态，不管 PB4 的方向寄存器的设置，并且具有输出比较功能；该位为“0”，则表示 PB4 作为普通双向 I/O 使用。

3.5.3 MC68HC05J3 的 16 位定时器

定时器是单片机内部不可缺少的功能部件。J3 系列的内部除了具备其他 J 系列内部通用的 15 级多功能定时器外，还具有一个 16 位定时器。该定时器功能强大，除了具有定时功能外，还可实现输入捕捉、输出比较功能。利用这些功能，可以完成计时、计数、脉冲频率测量、脉冲宽度检测，也可以使单片机非常容易地产生各种频率的控制波形。这些功能是在具体应用中经常要使用到的。

一、J3 系列内部 16 位定时器结构

该定时器的结构如图 3-26 所示。该系统具有多个寄存器，其中包括三个 16 位寄存器：自由运行计数器、输入捕捉寄存器、输出比较寄存器，还包括一个 8 位控制寄存器和一个 8 位状

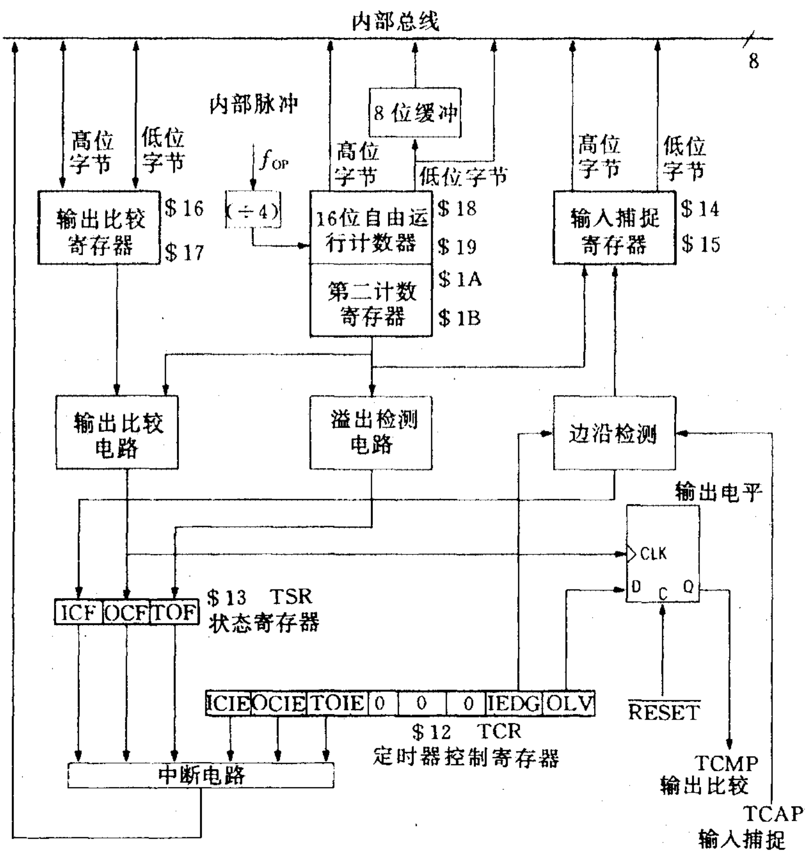


图 3-26 J3 的 16 位定时器

态寄存器。外部有两个引脚:捕捉功能引脚 TCAP 及比较功能输出引脚 TCMP,这两个引脚与 PB 端口的 PB4,PB5 共用。

16 位定时器的主体是一个 16 位运行计数器,其输入是内部时钟的 4 分频信号。该计数器自由运行,除复位外,不受其他任何因素的影响;它可读但不可改写。自由运行计数器一方面可以产生溢出中断信号,另一方面又作为整个定时器系统的参考计数器,并在此基础上实现输入捕捉功能和输出比较功能。

输入捕捉功能的实现,是当该定时器输入引脚 TCAP 上产生设定的跳变(上升沿或下降沿)时,系统可及时将其计数器的计数值自动装入到输入捕捉寄存器中,用户通过读取该数值可得知该跳变事件产生的时刻。利用这一特性,用户可轻易编程,实现对外部脉冲信号宽度、周期等性能的检测。

输出比较功能的实现,体现在当自由运行计数器的值累加到等于输出比较寄存器的值时,可以自动、及时地在定时器输出脚(TCMP)上输出设定的电平信号。用户可以利用这一特性,准确地控制输出引脚(TCMP)在未来特定时刻输出指定的电平信号,从而达到产生各种控制信号波形的目的。

二、自由运行计数器

该计数器为 16 位,自由运行,不为任何指令控制。单片机复位完毕后即进行累加计数。计数过程中可以随时读取其计数值,但不可改写或预置。其输入计数脉冲为内部时钟的 4 分频信号。如果使用晶振频率为 4 MHz 的晶振,则其计数脉冲宽度为 2 μs。

可以从两个不同地址的寄存器中读取计数器的数值,18H,19H 两单元称为第一计数器,1AH,1BH 两单元称为第二计数器。高 8 位数据存放在低地址单元中。在读取任何一个计数器的值时,要注意读高、低字节的顺序。正确顺序应该是先读高 8 位数据,再读低 8 位数据。这是由于计数器是 16 位的,而 CPU 为 8 位,不能同时读取两字节内容,只能用多条指令完成读 16 位数据的操作。为了正确反映读取的内容,所读高、低字节应是同一时刻的数值。为此,在该定时器中有硬件部分保证这一功能的实现。当读取高字节时,系统会同时把低字节的数值送入一缓冲器中,读低字节时的数据则由该缓冲器提供。这样就保证了高、低字节的内容是同一时刻的。另外,一旦读了高字节,就一定要读低字节,不然缓冲器的内容就不会更新,即使进行多次读高字节都是这样。

第一计数器与第二计数器的区别是,第一计数器内容的读取会对状态寄存器的溢出标志 TOF 清“0”,而读第二计数器操作则不会影响 TOF。标志 TOF 是在计数器发出溢出时置位,即计数值从 FFFFH 变为 0000H 时置“1”。如果允许该中断产生,且中断屏蔽位无效,则可以产生溢出中断,中断程序入口地址存放在 0FF2H,0FFF3H 中。

四、输入捕捉寄存器

输入捕捉功能的实现,是由引脚 TCAP 上发生软件设定的跳变(上升沿或下降沿)时激发,此时系统会及时将自由运行计数器的值自动装入到输入捕捉寄存器中,从该寄存器中可以读得跳变事件产生的准确时刻。

该寄存器也是 16 位的,但分二个单元存储,单元 14H 存放高字节,单元 15H 存放低字节。其内容只可读不可改写。该寄存器一旦捕捉到计数值,便会在定时器的状态寄存器上建立相应标志。如果在控制寄存器中允许输入捕捉中断,且系统已开放中断,则可以产生输入捕捉中断。中断程序入口地址存放在 0FF6H 和 0FF7H 两单元中。

在读入捕捉数据时，其高、低字节读入顺序是很严格的，应先读高字节(14H)，再读低字节(15H)。当读入高字节后，系统会自动禁止捕捉功能，防止在未读入低字节前又进行第二次捕捉，直到低字节读入后才又允许。因此，先读高字节，然后马上读低字节，这样才能准确读入数据，而又不影响下次捕捉功能的实现。

利用该功能可以实现对频率或周期的测量。例如，第一次捕捉值为 n_1，第二次捕捉值为 n_2，则这两次事件产生的时间间隔可以计算为：$(n_2-n_1)\times 4\times 2\times \frac{1}{f_{OSC}}$。其中，$f_{OSC}$为外部所接振荡器的频率。

五、输出比较寄存器

输出比较寄存器也为16位。它由两个8位寄存器组成，高8位数据存在16H中，低8位数据存在17H中。该寄存器不受定时器硬件与复位的影响，可读可写。如果不使用定时器的输出比较功能，则这两个字节可用于存储用户数据。

输出比较功能通过该寄存器实现。该寄存器存入的内容代表了未来某时刻。当达到该时刻，也就是说自由运行计数器的计数值等于该寄存器的内容时，在引脚 TCMP 上会及时输出相应电平。输出电平的高低可以通过定时器的控制寄存器(12H 单元)选择。此时，如果置相应的中断允许位有效，则会产生相应中断，其中断入口地址存放在 0FF4H，0FF5H 单元，以便程序及时修改内容，确定下次产生的时刻以及输出的电平。

当向比较寄存器的高字节写入数据后，在写入低字节前会禁止该比较功能，以避免在更新高、低字节内容时产生错误的比较输出。因此，用户仍要注意更新高、低字节的顺序，应是先高字节后低字节，写了高字节就一定要写低字节，不管低字节是否要改变。

利用该功能可以实现多种控制功能，产生某种控制波形以及各种时间计时信号等。

六、定时器控制寄存器

该定时器的控制是由一个8位控制寄存器实现的，如图 3-27 所示。它决定了该定时器的有关功能，包括溢出中断、输出比较中断、输入捕捉中断的允许与禁止以及输出比较电平选择和输入捕捉触发电平方式选择。

地址	B7	B6	B5	B4	B3	B2	B1	B0
12H	ICIE	OCIE	TOIE	0	0	0	IEDG	OLV

图 3-27　定时器控制寄存器

ICIE——输入捕捉中断允许控制。为“1”允许，为“0”则禁止。

OCIE——输出比较中断允许控制。为“1”允许，为“0”则禁止。

TOIE——计数器溢出中断允许控制。为“1”允许，为“0”禁止。

IEDG——输入捕捉激发方式选择。该位为“1”表示引脚 TCAP 的上升沿激发，为“0”表示引脚 TCAP 的下降沿激发。

OLV——输出比较功能输出电平选择。为“1”表示在 TCMP 上输出高电平，为“0”表示输出低电平。

第四章　廉价单片机的开发系统

J,K 系列与其他 HC05 系列类似,有多种开发手段与开发工具。可以是高档的 HDS200 系统、MMDS05 系统;中档的 HC05/08 MMEVS 开发系统、HC05 的 EVS 开发系统和 EVM 系统;也可以选择低档的、价格便宜的开发工具,例如,编程器、软件仿真器等。目前大家普遍使用的是 HC05EVS 开发系统和软件仿真器与编程器。本章将对这两种开发系统作详细介绍。

4.1　MC68HC05EVS 开发系统

J 系列的 EVS 开发系统是在 MC68HC05EVS 仿真系统的基础上建立起来的。HC05EVS 开发系统为 MC68HC05 各系列单片机提供了开发的手段。该系统经济实用、功能强大,可以完成多种仿真与调试工作,能够大大提高开发的效率。

HC05EVS 开发器由两块电路板构成,如图 4-1 所示。两电路板叠放在一起,下层板为基板,是各系列开发系统所共用的;上层板则根据各系列不同而有所不同。

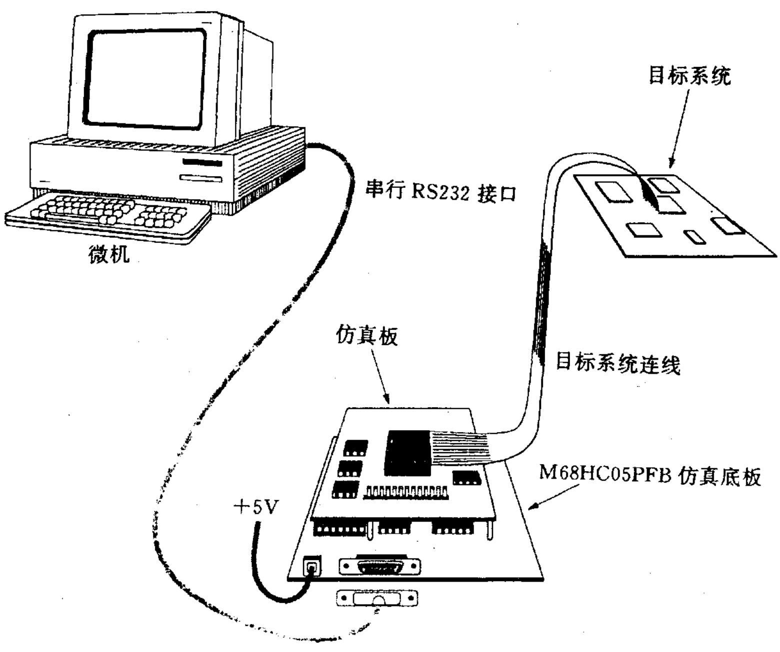


图 4-1　HC05EVS 开发系统

4.1.1　基板 PFB 的结构

EVS 的基板为 HC05 各系列开发工具所共用,主要由通信端口、虚拟内存、监控程序、总线控制电路以及有关用户按键与跳线组成。如图 4-2 所示。

其中,端口 P2 为通信端口,用以连接微机。用户的调试命令、用户程序都是从这个通信端

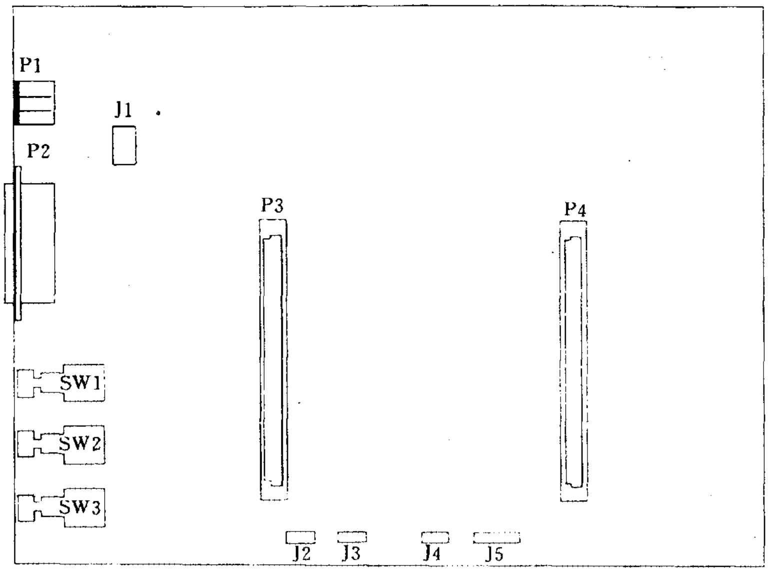


图 4-2 HC05EVS 基板

口传送到该开发器中的；同时，开发器的状态，包括 CPU 寄存器、用户 RAM、单片机并行 I/O 的状态等等也通过这个端口传送到微机上显示。该端口为 D 型 25 芯插座，是标准 RS232 串行通信端口。用户只要用电缆与微机 RS232 的串口 COM1 或 COM2 连接即可。所用的电缆要与微机串行通信口各信号线一一对应，即 TXD 对应 TXD，RXD 对应 RXD 等等；不能使用内部已交叉连接的通信电缆线。如果微机的串口为 9 芯端口，则要使用标准 9 芯-25 芯或 25 芯-9 芯转换插口进行转换。

P1 为开发器的直流稳压电源输入端，包括＋5V 系统使用电压以及灌写程序所使用的 V_{PP}编程电压。在调试时，V_{PP}电源可以不接入，在灌写程序时才接通。V_{PP}编程电压的大小因 HC05 系列的不同而有所区别。

P3，P4 为上层板与下层板实现数据传送与控制的连接接口。

J1 跳线用来设置所仿真的虚拟 ROM 的大小。J2 在工厂测试时使用，用户使用时，该跳线要设置好。J3 为目标复位跳线，当该跳线设置时，将用户的目标系统的复位线与开发器的复位线连接在一起。J4 为 C/D 选择跳线，对于 J 系列，该跳线应设为空。J5 跳线用于写保护，设置有效后，用户的虚拟 ROM 便具有了写保护功能。

该基板上共有 3 个按键。其中，SW1 为中止用户程序执行按键，一旦按下，即立即中止用户程序，返回监控程序，并显示目前单片机运行的状态。SW2 为用户复位按键，当该键按下时，开发器立即复位单片机，并马上执行用户程序。要实现该功能，用户程序的复位应该已经设置好，否则无法执行用户程序，并造成死机。SW3 为主复位键，当按下该键后，开发器实现复位，执行监控程序，等待用户调试命令。该按键常用于死机的退出或用户程序的退出。

4.1.2 MC68HC05J3 仿真模块

J 系列中以 J3 的功能最强。它不但内存大，而且具有 16 位多功能定时器，J 系列的其他功

能它都具备。该仿真模块不但可以开发 J3 系列单片机，还可以开发 J 系列中其他单片机。但要注意，在开发时各单片机的内存地址不同，还要关掉有关 J3 的特殊功能。该仿真模块同上面所讲述的 J1A 仿真模块类似，也同样可以与 EVS 开发系统的基板相连组成 J3 的 EVS 开发系统，或者与 MMDS05 系统相连组成 J3 的 MMDS05 开发系统。

该仿真模块只是具备仿真功能，并不具备灌写程序的编程功能。

一、J3 仿真模块的结构

J3 系列仿真模块(J3EM)的结构如图 4-3 所示，主要由 J3 的单片机、相关控制电路、连接目标系统的端口、连接逻辑分析仪的端口以及连接 EVS 基板的端口等组成。

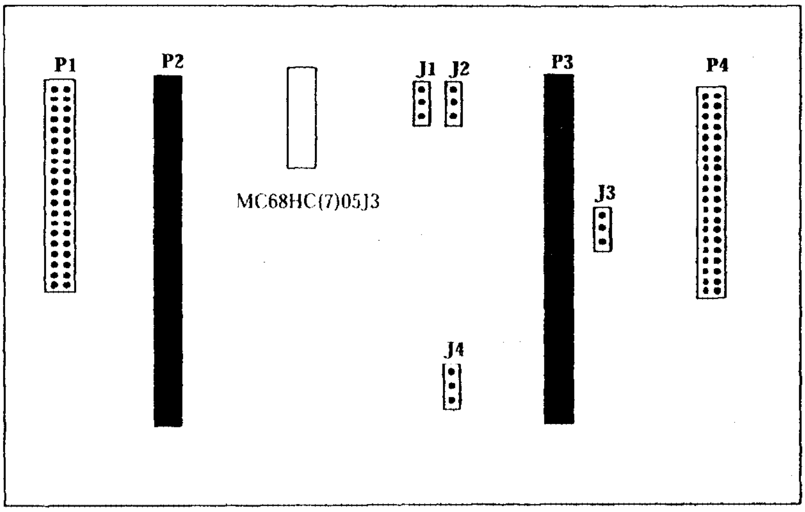


图 4-3 J3EM 结构

P4 为连接目标系统的端口，该端口各引脚的定义如图 4-4 所示。用户只要使用 20 芯的扁平线加转换端口，使仿真器与目标系统相连即可。注意使双方引脚一一对应。该 P4 端口输入/输出信号完全仿真实际 J3 单片机的输入/输出信号。

P2，P3 是该模块与 EVS 基板相连的端口。通过该端口，可以直接使该 J3EM 板与基板相连，也可以实现与 MMDS05 开发系统的连接。

P1 端口为逻辑分析仪端口。通过该端口，在逻辑分析仪上可以观察分析仿真效果。在该端口的各引脚上直接反映了单片机 CPU 的操作过程，反映出每时刻总线操作的情况。这对我们分析和解决问题，加快调试过程有很大的帮助。

二、跳线的设置

J1，J2 为 J3 仿真板振荡器的选择跳线。如果振荡信号来源于仿真板本身，则 J1，J2 的跳线应设置为 1-2 短

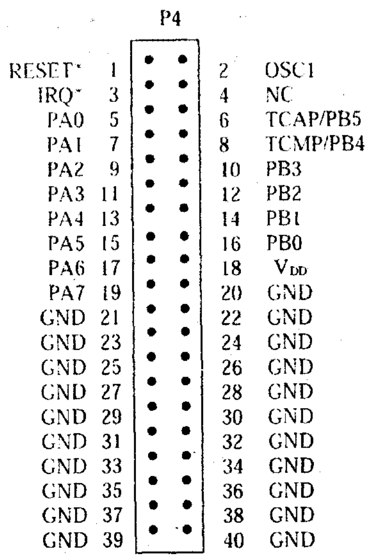


图 4-4 J3EM 板 P4

接;如果来源于外部,则 J1,J2 的跳线应设置为 2-3 短接。

J3 为复位信号选择跳线。当设置为 1-2 短接时,表示复位信号来源于开发器本身;而当设置为 2-3 短接时,表示用户目标系统的复位引脚与开发器的复位引脚相连,这样,目标系统的复位操作也会引起开发器的复位操作。

4.1.3 MC68HC05J 系列 HC05EVS 开发系统的软件

EVS 开发系统的软件系统主要包括汇编器与调试软件,用来提供汇编程序的编辑、汇编、调试等功能。

汇编器主要使用 CASM05.EXE 或 IASM05.EXE,调试软件为 EVM05.EXE,另外还有集成开发软件 RAPID.EXE。

以上各软件都是运行在 MS-DOS 环境下。

一、CASM05 汇编器

该软件主要用于汇编程序的编译,产生目标文件、列表文件和图表文件。

1. 目标文件

它是以 S 记录格式表示的机器码文件,其扩展名为 .S19。该 S 记录格式是 MOTOROLA 公司制定的十六进制目标文件格式,它把目标代码与定位地址用可打印的 ASCII 码字符串表示。该字符串包含地址、数据以及校验代码,表示如下:

类 型	记录长度	地 址	数 据	校验和

表 4-1 给出了以上各个域的构成和内容说明。

每个 S 记录都可用 CR/LF/NUL 来终止,若干行 S 记录构成一个 S 记录模块。在操作系统中,都把 S 记录文件作为通常的 ASCII 码文本文件来看待。

传送 S 记录文件时,可用其记录长度和校验和区域的内容来验证传送的准确性。

为了满足各种编码、传送及解码功能的需要,定义了 8 种 S 记录类型。各类型说明如下:

S0——每个 S 记录文件的头记录,其程序/数据域中可包含对后面的 S 记录块的任意说明信息,其地址域通常为 0。

表 4-1 S 记录各域说明

域	可打印字符数	内 容
类型	2	S 记录类型——S0,S1 等。
记录长度	2	记录中所含字符对的数目。其中,类型和记录长度字符不包括在内。
地址	4,6 或 8	数据域中的数据将装入到存储器的地址,它可以是 2 个、3 个或 4 个字节。
程序/数据	0~n	由 0 到 n 个字节的可执行程序(目标程序)或数据组成。有些程序限制这个域的字节数必须少于 28 个字节。
校验和	2	是组成记录长度、地址和程序/数据的所有字符之和的反码的低位字节。

S1——该记录包含程序/数据和存储其程序/数据的存储器的 2 字节地址。

S2——该记录包含程序/数据和存储其程序/数据的存储器的 3 字节地址。

S3——该记录包含程序/数据和存储其程序/数据的存储器的 4 字节地址。

S5——该记录包含在一个特定记录块中被传送 S1,S2 和 S3 记录的数目,这个数目将出现在地址域上。该记录没有程序/数据域。

S7——S3 记录块的结束记录。该记录没有程序/数据域,地址域可包含控制将转向的指令的 4 字节地址。

S8——S2 记录块的结束记录。该记录没有程序/数据域,地址域可包含控制将转向的指令的 3 字节地址。

S9——S1 记录块的结束记录。该记录没有程序/数据域,地址域部分可包含控制将转向的指令的 2 字节地址。

如果用户的系统不支持. S19 数据格式,也可以由 CASM05. EXE 产生扩展名为. HEX 的文件。该数据格式为 Intel 公司、Zilog 公司等所采用,也是一种通用的数据存储文件格式。如要产生这种. HEX 文件格式,可按如下格式执行 CASM05 程序:

C>CASM05 *file*. ASM　H

2. 列表文件(扩展名为. LST)

它由源程序和目标代码、指令周期、地址等组成,常作为用户最后的存档文件或者调试过程中用户所使用的源程序文件。由于该文件清晰地列出了各条指令的执行地址,因此便于调试程序。要产生该列表文件,必须在执行 CASM05 程序的命令行中增加 L 选项,即

C>CASM05　*file*. ASM　L

3. 图表文件(扩展名为. MAP)

该文件包含了源文件所使用符号的有关信息。如果在调试中使用源码程序调试,即在调试窗口中是以源程序形式显示,能显示出用户所定义的符号。为此,在调试时,除了 S 记录目标文件,还要将该. MAP 文件下载到开发器中。

4. 汇编指示命令

用户编写源程序一定要按照规定的指令格式,CASM05 还支持一些高级汇编指示命令,用于控制汇编过程和汇编方式。汇编指示命令不是 MC68HC05 指令助记符,在引用该命令时必须在命令行的开头以"#"或"$"作为引号符,以示区别。汇编指示命令如表 4-2 所示。

下面分别对以上汇编指示命令进行说明。

BASE 指示命令

BASE 汇编指示命令是用于改变现行文件缺省基数标记的数。命令后的参数必须是现行的基数及其标记。其命令格式及其应用举例如下:

```
$BASE   10T     ;改变缺省基数标记的数为十进制数。
$BASE   0AH     ;改变缺省基数标记的数为十进制数。
```

由 BASE 指示命令修改后所得的新的缺省标记的基数将在其后的文件中一直有效,直至文件结束或遇到另一个 BASE 指示命令为止。如果不能确信现行的缺省基数是什么数时,最好先用一个 BASE 命令来指定一个明确的基数形式。

CYCLE ADDER 指示命令

在汇编器中含有一个指令周期的内部计数器,称为周期加法器。这是由两个汇编指示命令 CYCLE-ADDER-ON 和 CYCLE-ADDER-OFF 来进行控制的。

当汇编器收到允许计数器指示 CYCLE-ADDER-ON 时,先清内部的周期加法器,然后使

其周期加法器开始对随后被汇编的指令的周期数进行求和，直至当汇编器收到关闭计数器指示 CYCLE-ADDER-OFF 时，才禁止周期加法器工作，并将当前周期加法器的值写入列表(.LST)文件中。

表 4-2　汇编指示命令

指示命令	操　　作
BASE^	改变现行缺省基数标记的数为二、八、十或十六进制数。
CYCLE-ADDER-OFF	停止内部计数器累加指令的周期数，并输出计数器的计数值到.LST 文件。
CYCLE-ADDER-ON	允许内部计数器开始累加指令的周期数。
ELSEIF	条件汇编中与 ENDIF 配对的中间语句。
ENDIF	条件汇编的结束语句。
IF^	如果条件"∧"为真，则汇编指定的代码段。
IFNOT^	如果条件"∧"为假，则汇编指定的代码段。
INCLUDE '∧'	在源文件中，把链接的附加文件命名为"∧"。
MACRO^	建立一个名字为"∧"的宏指令。
MACROEND	结束一个宏定义。
SET^	设置由"∧"命名的条件为真。
SETNOT^	设置由"∧"命名的条件为假。

注意，为了执行 CYCLE ADDER 指示命令，必须把在"Assemble"子菜单中的 Listing 和 Cycle Cntr 选项设置为"ON"状态。

条件汇编

汇编器允许用条件汇编指示来对满足条件的程序块进行汇编，其他的则被忽略。

条件汇编指示语句的格式如下：

```
$SET (或 SETNOT)  参数
$IF (或 IFNOT)  参数
    [程序块 1]
$ELSEIF
    [程序块 2]
$ENDIF
```

其中，SET 指示命令是设置它的参数值为真，而 SETNOT 指示命令是设置其参数值为假。由 SET 或 SETNOT 所设置的参数值最大为 25。程序块 1 是 IF 语句中的参数值为真时或 IFNOT 的参数值为假时，被汇编的代码段；程序块 2 则刚好相反，它在 IF 的参数值为假或 IFNOT 的参数值为真时，被汇编的代码段。下面给出一个条件汇编的例子。

```
$SET        debug       ;sets debug=true
$SETNOT     test        ;sets test=false
            nop         ;always assembles
```

```
            nop             ;always assembles
$IF         debug           ;if debug=true
            jmp start       ;assembles
$ELSEIF                     ;if debug=true
            jmp end         ;does not assemble
$ENDIF;
            nop             ;always assembles
            nop             ;always assembles
$IF         test            ;if test=true
            jmp test        ;does not assemble
$ENDIF;
```

INCLUDE 指示命令

INCLUDE 是一个附加文件的链接指示命令，利用它，可以把另一个源文件插入当前的源文件一起汇编，成为一个完整的源程序。

格式：$INCLUDE "［驱动器名:］［路径名］文件名"

其中，文件名前的驱动器名、路径名为任选项，如果文件在当前目录中，则可将它们略去。

汇编开始时，被链接文件一起与源程序处理，当遇到文件结束伪指令 END 时，汇编会停止运行。注意，被链接的文件中不能包括 END 伪指令，END 只能出现在主程序中。汇编结束后，在其所得的列表(.LST)文件中是否显示被链接文件的源代码，取决于 Assemble 子菜单中"INCLUDE"选项的设置。

宏处理 MACRO

在源程序中，若有一段程序需要多次使用，为避免重复书写这样一段程序，可用宏定义把需要重复出现的程序块定义成宏指令，此后在宏指令出现的地方，宏汇编程序总是自动地把它们替换成相应的代码段。可见，宏指令类似于用 INCLUDE 引用文件，但它比 INCLUDE 文件更复杂。例如，宏指令定义中可使用标号，并且可以接收参数值。

宏定义格式：$MACRO　宏指令名 [形式参数,……]

在宏指令定义中，使用了"形式参数"，它们在引用宏指令时被给出的一些名字或数值(实在参数)所取代。使用形式参数给宏指令带来很大的灵活性。下面程序段定义了一个延时的宏指令。

```
$MACRO delay count
        mov a, %1      ;其中%1用于接收宏调用时的实参值。
LOOP:   DECA
        BNE LOOP
$MACRO END
```

宏调用格式：宏指令名 [实在参数,……]

例如，对以上延时宏指令的调用可表示为

delay 100 T　　;表示调用 LOOP 循环 100 次。

说明：

(1) 实在参数将对应地替换宏指令中的形式参数，如果实参的个数比形参多，则被汇编器忽略；但如果实参的个数比形参少，则汇编器将给出一个错误信息。

(2) 在汇编过程中，任何一个宏指令内部定义的标号都被认为是局限于该宏指令。为避免同一宏指令多次使用重复标号的问题，在每次宏指令扩展时，任一局部标号通过加一个唯一的四位数字到原始的标号名尾部，使其原标号被改变成一个唯一的形式，从而避免了源程序中标号重复的问题。因此，在此规定，任一宏指令内部定义的标号长度不能超过 10 个字符。

(3) 程序的执行不能从外部跳入宏指令内部，但允许从宏指令内部跳出。

(4) 从列表文件(.LST)程序清单中是否可看到宏指令调用所产生的相应代码，取决于对 Assemble 子菜单中的“MACRO ”选项的设置。

5. 伪操作指令

在 IASM05 汇编器中，允许使用的伪操作指令如表 4-3 所示。

表 4-3　伪操作指令

伪指令	操　作　说　明
RMB *n* 或 DS *n*	定义一个存储区，其中预留 *n* 个字节。这里 *n* 可为数字或标号。
FCB *m* 或 DB *m*	定义一个字节所包含的值。这里 *m* 可为标号、数字或字符串。当参数 *m* 不是字符串时，每个参数赋给一个字节，多个参数时必须用“,”隔开。如果参数 *m* 为字符串时，将用多个字节存储其对应的 ASCII 码。
FDB *m* 或 DW *m*	定义一个字(16 位)的存储内容。这里 *m* 可为标号、数字或字符串，但每一个参数占 16 位，多个参数时必须用“,”隔开。
LAB:EQU *n*	把数字或标号 *n* 的值赋给标号 LAB。
ORG *n*	设置或改变程序计数器的值。*n* 可为标号或数字。
LAB:RAMLOC *n*	定义 RAM 位置。这里 *n* 可为数字或标号。

这些伪操作指令中，RMB 常用来定义变量、缓冲区，而 FCB 则可定义变量、填充缓冲区数据，FDB 伪指令常用来定义一个字长(两个字节)的变量或用作中断向量、复位向量的设置。

二、EVM05.EXE 调试软件

当微机通过串行口与 EVS05 开发器连接后，用户就可以使用 EVM05.EXE 软件进行调试工作了。在这个软件环境里，用户所做的调试工作可以有很多，包括程序的单步执行，断点的设置，内存内容的查看与修改，程序的修改，直接对 I/O 端口的输入/输出操作，直接对 CPU 的累加器 A、变址寄存器 X、状态寄存器 CCR、堆栈指针 SP、程序计数器 PC 的操作。关于其详细命令的解释，详见表 4-4。

表 4-4　EVM05 调试命令

命　令	解　释	举　例
A	设置累加器 A 的内容	A　10
ASM	修改程序	ASM　100(地址)
CCR	设置状态寄存器	CCR　00
CLEARMAP	清除符号	CLEARMAP
EXIT	退出 EVM05	EXIT
G	快速执行程序	G 或 G　100(地址)
HELP	显示帮助菜单	HELP

续表 4-4

命　令	解　释	举　例
LOAD	装载.S19 文件	LOAD
LOADMAP	装载.MAP 文件	LOADMAP
LOADALL	装.S19 和.MAP 文件	LOADALL
MD	显示内存内容	MD 50(地址)
MM	内存单元内容修改	MM　50(地址)
PC	设置程序计数器	PC　100
T	单步执行程序	T 或 T5(单步执行 5 次)
X	设置 X 寄存器的值	X 30
B1,B2,B3,B4	断点设置	B1　200(地址)
PRTA	设置 PA 端口输出数据	PRTA　55
PRTB	设置 PB 端口输出数据	PRTB　50
PRTC	设置 PC 端口输出数据	PRTC　40
PRTD	设置 PD 端口输出数据	PRTD　60
DDRA	设置 A 端口方向寄存器内容	DDRA　FF
DDRB	设置 B 端口方向寄存器内容	DDRB　00
DDRC	设置 C 端口方向寄存器内容	DDRC　55

该软件在 DOS 环境下运行。在 DOS 提示符下执行“EVM05　2”或“EVM05　1”便可进入 EVM05 多窗口调试环境，如图 4-5 所示。从这些窗口中可以看到 CPU 状态，I/O 端口内容，堆栈内容，RAM 内容，断点设置，程序执行位置等信息。该软件可以被其他集成软件调用，从而进入 EVM05 调试环境。例如，在下述的 Rapid. EXE 软件中所具有的调试功能，是 Rapid 软件通过调用 EVM05. EXE 程序而实现的。

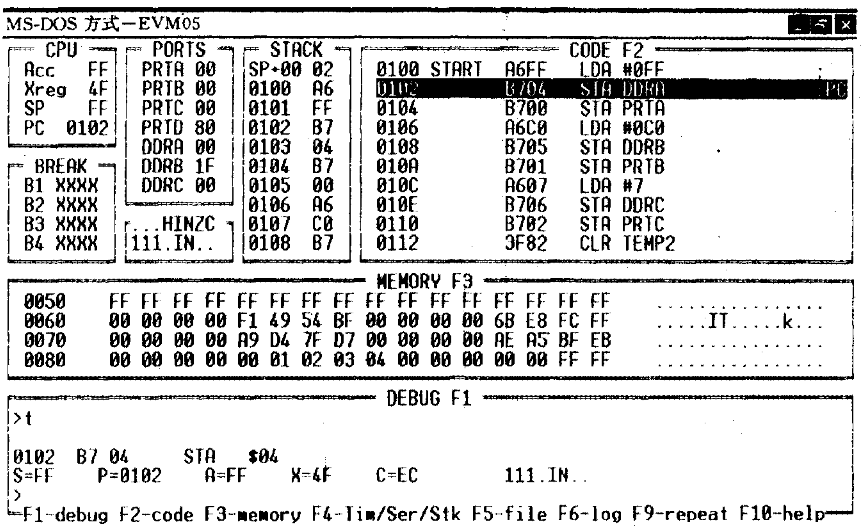


图 4-5　EVM05 调试窗口

三、Rapid 集成开发环境

Rapid 软件是专门针对 MOTOROLA 开发系统设计的软件开发工具。由于 MOTOROLA

的单片机种类很多，除了 HC05 系列外，还有 HC11，HC08 等多种档次的单片机，由于它们之间性能相差很大，其汇编器和调试软件都有一定区别。为了使用户操作方便，Rapid 软件可以将各种软件集成在一起。例如，编辑器、交叉汇编器、通信软件、编程器、仿真开发器、调试器、C 语言编译器等。用户只要在 Rapid 环境下，按热键即可完成汇编、编译、仿真调试等工作。这些工作的实现是 Rapid 自动调用相应软件实现的。用户通过安装 Rapid 的集成环境，可以建立自己的单片机开发环境。这样，用户可以在 DOS 下大大减少操作量，可以自由地在编辑器、汇编器、仿真器等之间快速切换，加快调试过程。

在使用 EVS05 开发系统时，同样可以使用 Rapid 的集成环境来实现汇编程序的编辑、汇编、调试等工作。此时，常将 CASM05. EXE 与 EVM05. EXE 集成在 Rapid 环境中，如图 4-6 所示。这样，就可以在 Rapid 环境下编辑、修改源程序；按功能键 F4，自动调用 CASM05. EXE 软件汇编源程序，生成. S19 文件、. MAP 文件和. LST 文件；按功能键 F6，可以自动调用 EVM05. EXE 软件而进入调试环境，调试完毕，在 EVM05 中输入 EXIT 命令又可返回 Rapid 环境。

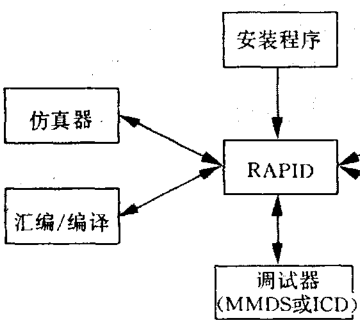


图 4-6　EVS 开发系统 Rapid 集成环境

在各功能键中，F1 为帮助键，它将启动显示帮助文件；F2 为存盘命令热键；F3 为装载源程序热键；F5 为退出 Rapid 环境热键；F9 为进入 MS-DOS 热键。用户可以通过 Rapid 的安装程序 Rinstall. EXE 安装所集成的软件，并定义各个热键如 F4，F6，Alt＋F4，Ctrl＋F4，F7，Shift＋F5 等的激活功能。

4.2　MMEVS05/08 开发系统

MMEVS 开发器是 MOTOROLA 公司开发的一种新型开发系统。它不但支持 MC68HC05 系列，还支持另一种新系列 MC68HC08。它是在 EVS 基础上发展起来的，其整体框架结构与 EVS 类似，也是由上下两层板构成，底板为各系列单片机开发时的基础，上层板因所开发的单片机系列不同而有所不同。同样，称上层板为仿真模块，下层板为基板。

该 MMEVS 开发系统主要特性如下：

(1) 实时在线仿真；

(2) 可以支持最大 64 KB ROM 空间；

(3) 可任意设置 64 个断点；

(4) 软件命令选择振荡时钟频率；

(5) 具有批处理命令，自动完成用户定义的一系列命令；

(6) 完美的源程序符号调试功能；

(7) 与微机的串行通信波特率最大可达 57 600 bps；

(8) 软件命令复位、中止功能，一切操作都在微机上操作完成。

一、MMEVS 基板结构

MMEVS 基板虽然功能增多，但其板上跳线与按钮减少了。如图 4-7 所示，有一个直流电源输入端，一个与微机连接的 RS232 串行通信端口以及与上层仿真模块的连接插头。按键方

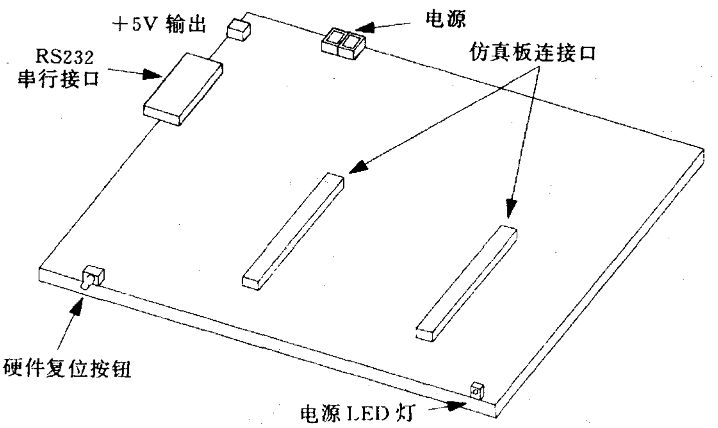


图 4-7 MMEVS 基板

面只有一个硬件复位按钮。这种复位按钮不会像 EVS 基板上的复位键一样在调试中常用，只是在与微机联机出现问题时才使用；而其他调试过程中的复位，完全可以通过微机发出复位命令完成。图中还可见一个+5 V 输出端口，这是该板的+5 V 电源输出端口，为其他设备如编程板、目标系统提供+5 V 电源。

在 MMEVS 基板上，共有 4 个跳线以进行不同的设置。J1 为工厂测试基板性能所使用，平时使用时应该将跳线设置为 1-2 短接。

J2，J3，J4 根据上层仿真板上仿真单片机的类型而设置，仿真 PA，PB，PC，PD 端口电压基准。其中，J2，J3 分别设置 PA，PB 端口；PC，PD 端口的设置由 J4 完成。如果这些跳线设置为 1-2短接，则表示各端口工作在+5 V 电压；如果上层板仿真的单片机能够工作在低电压下(例如 3.0 V)，则这些跳线设置为 2-3 短接。

二、MMEVS 软件系统

该系统的软件同 EVS 类似，也包括交叉汇编器与调试工具，同样是用 Rapid.EXE 建立起来的集成开发环境。

它的交叉汇编器除了用于 68HC05 系列的 CASM05.EXE 外，还有用于 68HC08 系列的 CASM08.EXE。对于 J，K 系列，使用 CASM05.EXE 进行汇编，生成 .MAP、.S19 和 .LST 文件。这时所使用的交叉汇编程序跟上节中所述的汇编器的一样。

调试工具软件跟 EVS 系统不同，主要包括以下几个程序，见表 4-5。

这里要注意的是，扩展名为.MEM 的文件在使用这套系统时一定要具有与上层仿真模块相对应的数据文件。MMEVS05(或 MMEVS08)在运行时首先要读取该数据文件，读取过程是

软件自动完成的，自动搜索相配套的文件。如果在当前目录中搜索不到，它会自动提问搜索路径，如始终无法找到，那么系统将无法继续运行。

表 4-5　调试工具软件

程序名称	程序说明
MMEVS05. EXE	MMEVS05 调试工具(适用于 MS-DOS)
MMEVS05X. EXE	MMEVS05 调试工具(适用于 Windows 下 DOS 窗口)
MEVS05VX. HLP	MMEVS05 帮助文件
MMEVS08	MMEVS08 调试工具(适用于 MS-DOS)
MMEVS08X	MMEVS08 调试工具(适合 Windows 下的 DOS 窗口)
MEVS08VX. HLP	MMEVS08 帮助文件
*.MEM	上层仿真模块的数据文件

MMEVS05. EXE 或 MMEVS08. EXE 都可集成在 Rapid 环境下，只要通过 Rapid 安装程序 Rinstall 就可以实现。

在 DOS 下运行，见以下几例：

MMEVS05　　　　　　　　　　;采用缺省数运行。

MMEVS05　2　　　　　　　　;串行口为 COM2。

MMEVS05　BW　　　　　　　;单色显示器运行。

MMEVS05　－B　　　　　　　;设置通信波特率为 9 600 bps。

MMEVS05　〈*filename*〉　　;运行后自动装载对应的. S19 与 . MAP 文件。

MMEVS05 －M　〈××××.MEM〉 ;指定读取××××.MEM 数据文件。

当成功进入到 MMEVS05 的调试环境中后，就可以见到如图 4-8 所示的界面。

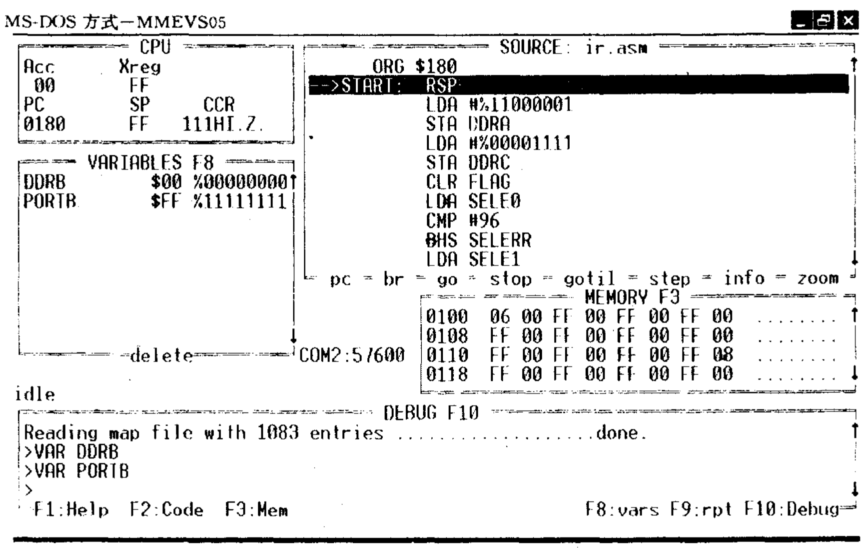


图 4-8　MMEVS05 调试环境

由图中可看出，MMEVS 调试环境与 EVS 的 EVM05 调试环境类似，但它比后者更加清晰、实用，主要包括 CPU 窗口、源程序窗口、内存数据窗口、用户调试命令输入窗口和变量窗

口，另外还有一些状态的指示行和帮助信息。

在 CPU 窗口里，可以看到累加器 A、变址寄存器 X、程序计数器 PC、堆栈指针 SP 和状态寄存器 CCR 的内容。

从源程序显示窗口可以清晰看到源程序以及程序执行到的位置。如果 .MAP 文件未装入到 MMEVS 中，则在该窗口里显示的不是源程序形式，而只能显示汇编的结果。在源程序窗口里，我们可以将反显光条移动到指定的位置并建立或消除断点，设置 PC 指针到相应位置，寻找指定的汇编语句等操作。这些操作除了使用箭头键或鼠标键外，还要配合 Alt 键，如表 4-6 所示。

表 4-6 Alt 组合键的功能

按　键	功　　能
Alt-B	在指定位置设置/消除断点
Alt-F	查找指定的字符串
Alt-L	重复查找
Alt-G	连续执行程序到指定位置
Alt-P	设置 PC 指针到指定的位置
Alt-M	显示指定的源程序模块

变量窗口用于显示指定变量在调试过程中数值的变化。在 MMEVS 中可以通过命令 VAR 选择跟踪 32 个变量，但由于窗口太小只能同时显示 11 个变量，通过按功能键 F8 可以进入该窗口里翻页显示或删除指定变量。

内存窗口用于显示内存 RAM，ROM 的数据，一次可以显示 32 个字节。通过按功能键 F3 可以进入该窗口，选择显示内存的区域。

调试窗口为用户输入调试命令的地方，同时也显示用户操作的有关信息。大部分的调试命令都在这里输入。按功能键 F10 可以进入到该窗口。有关调试命令如表 4-7 所示。这里的调试命令与 EVS 开发系统类似，同 EVS 比较，它增加了很多新的命令。表 4-7 中只列出了部分常用命令，其他未列出来的详见 MMEVS 技术说明书。

表 4-7 MMEVS 开发系统常用的调试命令

命　令	解　　释
A	设置累加器 A 的值
ASM	修改汇编语句
BAUD	设置通信波特率
BR	设置断点
C	设置进位
CCR	设置条件码寄存器
CLEARMAP	清除 .MAP 文件
DASM	反汇编
EXIT	退出 MMEVS 系统
G	开始执行程序
GOTIL	执行程序到指定位置
HELP	显示帮助文件

续表 4-7

命　令	解　　释
I	设置中断屏蔽位
LOAD	装载 .S19 数据文件
LOADMAP	装载 .MAP 文件
LOADMEM	装载指定的 .MEM 文件
MD	显示内存内容
MM	修改内存单元
NOBR	清除断点
OSC	设置振荡器时钟频率
PC	设置程序计数器
REG	显示寄存器的数值
RESET	复位仿真器
RESETGO	复位仿真器后执行用户程序
RESETIN	允许目标系统的复位信号进入仿真器
RESETOUT	允许仿真器的复位信号输出到目标系统
SHELL	进入 DOS 环境
SNAPSHOT	保存屏幕内容到指定文件
STACK	显示堆栈的内容
T	单步执行一条语句
VAR	显示指定变量
STOP	停止用户程序的执行
X	设置 X 寄存器的值
Z	设置零标志位

三、MC68HC05J1A 仿真模块

由于 MC68HC05 各系列的结构与性能相差较大，所以每种系列的单片机都有其对应的仿真模块，这些仿真模块与其他设备相连而组成对应单片机的开发系统。

J1A 系列单片机相应的仿真模块为 J1AEM。该模块可以与上面所讲的 MMEVS 基板连接，组成一个 J1A 系列单片机 MMEVS 开发系统；也可以与 MMDS05 系统连接，组成 J1A 的 MMDS05 开发系统。

但该仿真模块不具备灌写程序的编程功能。

1. J1A 仿真模块的结构

J1A 的仿真模块 J1AEM 的结构如图 4-9 所示，主要由 J1A 单片机、相关控制电路、连接目标系统的端口、连接逻辑分析仪的端口以及连接 EVS 基板的端口组成。

P4 为连接目标系统的端口，该端口各引脚的排列与实际 J1A 单片机引脚的排列相似。用户只要使用 20 芯的扁平线加转换端口，使仿真器与目标系统相连即可。连接时，要使该端口的引脚与用户目标系统板上的 J1A 插座的引脚相对应。

P2，P3 是同 MMEVS 系统基板相连的端口。通过该端口，可以直接将 J1AEM 板与基板相连，也可以通过它实现与 MMDS05 开发系统的连接。

P1 端口为逻辑分析仪端口。通过该端口，可以在逻辑分析仪上观察分析仿真效果，便于分析问题和解决问题，加快调试过程。如用户无逻辑分析仪，亦可以不使用。

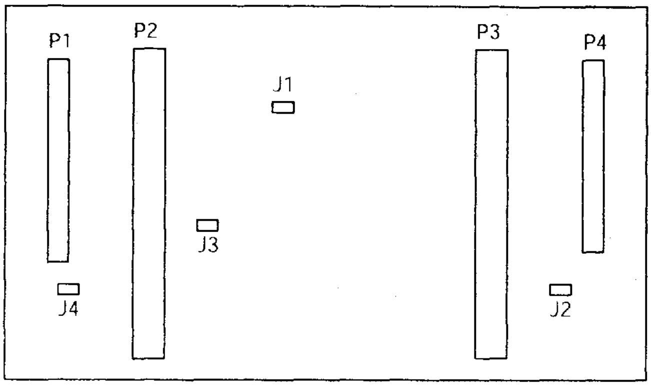


图 4-9　J1A 仿真模块

2. 跳线的设置

该板上共有 4 个跳线要设置，对这些跳线的设置可决定 J1A 仿真的有关硬件上的特性。

J1 跳线用以设置 PA 端口(PA0～PA7)、PB 端口(PB0,PB7)是否具有下拉电阻。跳线接通表示这些 I/O 引脚具有下拉电阻。下拉电阻的设置在键盘设计中是很有用的，一旦设置单片机内部下拉电阻有效，则用户系统不再需要增加下拉电阻。

J2 跳线用以设置 PA 端口(PA0～PA3)的中断功能。当该跳线设置为空时，允许该种中断；否则禁止。

J3 跳线用于单片机种类选择。该仿真模块除了仿真 J1A 外，还可以仿真 J2A 单片机。J2A 单片机性能与 J1A 系列相同，只是内存大小不同。跳线连接表示仿真 J1A 单片机。

J4 跳线用于仿真板振荡器的选择。该仿真模块的振荡信号可以来自模块本身的振荡器，亦可以来自开发系统控制基板。当 J4 设置为 1-2 短接时，表示使用模块板内的振荡信号；而设置为2-3短接时，则表示使用基板的振荡器。当使用 MMDS05 开发系统时，应设置为 2-3 短接。这样可以通过 MMDS05 开发系统的振荡频率选择，设置 J1A 的仿真振荡频率。

3. 软件系统

当用于 MMEVS 和 MMDS05 系统时，不再需要更多软件，只须将该模块的特征数据文件 00100V××.MEM(J1A)与 00101V××.MEM(J2A)通过 LOADMEM 装入到基板中去就可以了。

四、K 系列 MMEVS 仿真模块 M68EM05K3

对于 K 系列的仿真系统，可以使用基于 MMEVS 系统的 M68EM05K3 仿真模块。该模块可以与上述 MMEVS 的基板相接，也可以与 MMDS05 系统连接，以开发仿真 K 系列各种单片机，包括 MC68HC05K0，MC68HC705K1，MC68HC05K1，MC68HC05K3 四种芯片。

但该仿真模块不具备灌写程序的编程功能。

1. M68EM05K3 仿真模块的结构

M68EM05K3 仿真模块的结构如图 4-10 所示。主要由 K 系列单片机、相关控制电路、连接目标系统的端口、连接逻辑分析仪的端口以及基板连接端口等部件组成。

P4 为连接目标系统的端口，该端口各引脚的定义如图 4-11 所示。用户只要使用扁平线与转换端口，将仿真器与目标系统相连即可。

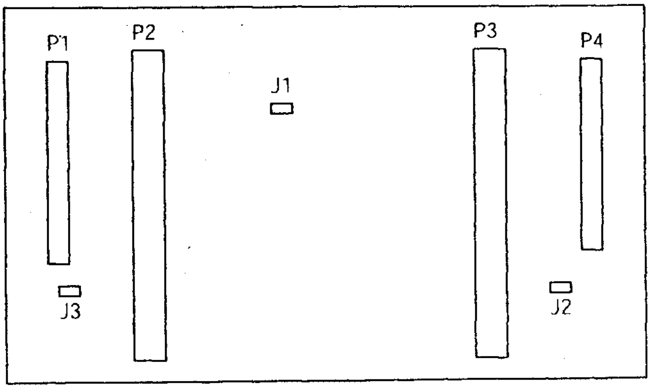


图 4-10　EMK3 模块结构

P2,P3 是同开发系统相连的端口。通过该端口可以直接使 EMK3 模块与 MMEVS 基板连接,或与 MMDS05 系统连接,构成 K 系列的开发系统。

P1 端口为逻辑分析仪端口。通过该端口可以使用分析仪来观察分析仿真效果,便于分析与解决问题,加快调试过程。

2. 跳线的设置

J1 跳线用于设置 PA 端口(PA0～PA7)与 PB 端口(PB0,PB1)是否具有下拉电阻。跳线接通表示这些 I/O 引脚具有下拉电阻。内部下拉电阻的设置是很有用的,一旦设置单片机内部下拉电阻有效,用户系统在单片机外便不再需要增加下拉电阻。

J2 跳线用以设置 PA 端口(PA0～PA3)的中断功能。当该跳线设置为空时,允许该种中断;否则禁止。

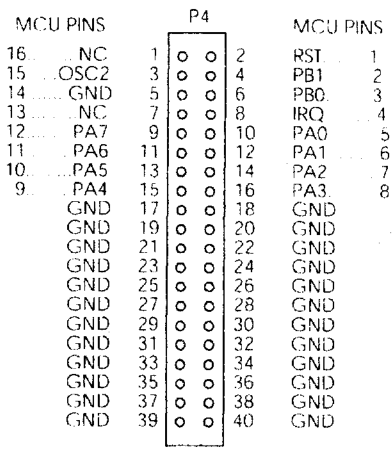


图 4-11　EMK3 板的 P4 接口

J3 跳线用于仿真板振荡器选择。该仿真模块的振荡信号可以来自模块本身的振荡器,亦可以来自开发系统的控制基板。当 J3 设置为 1-2 短接时,表示使用模块板内的振荡信号;设置为 2-3 短接时,表示使用基板上的振荡信号。

3. 软件系统

应用于 MMEVS 和 MMDS05 系统时,除了前述的软件外,这里不再需要更多的执行程序,只需将有关的 .MEM 数据文件(如表 4-8 所示)通过 LOADMEM 装入到基板中去,便可以利用调试程序来调试仿真了。

表 4-8　仿真不同系列单片机所使用的.MEM 数据文件

000K0V××.MEM	MC68HC05K0
000K1V××.MEM	MC68HC05K1,MC68HC705K1
00014V××.MEM	MC68HC05K3

4.3 J1A 的编程器与在线软件仿真器

EVS(或 MMEVS),MMDS05 开发系统调试功能完善,仿真效果真实,自然其价格上会比较高。如果用户不愿买这种开发工具,则可以考虑购买低档的开发工具,例如,软件仿真器与编程器。这一类仿真器简单、实用,但其仿真效果在实时性方面比较差。因为这类仿真器是软件仿真器,执行用户的程序时,是一条条语句解释执行的,因此,在时间方面不能仿真真正单片机运行时速度,运行速度要比实际运行时慢很多。虽然如此,这种开发器仍不失为一种较好的开发工具,它的调试命令非常丰富,也可以与用户目标系统相连,进行在线调试。

一、J1A 系列在线软件仿真器的特性

该仿真器用以调试 J1A 单片机系统,可以实现在线调试、片内 EPROM 编程等功能。它的主要性能如下所述:

(1) 经济实用的 J1A 单片机开发工具,实现在线调试;

(2) 使用 Rapid 集成软件环境,调试方便;

(3) 兼容 EVS 开发系统的调试命令;

(4) 具有断点设置,单步跟踪,修改内存,查看变量等源程序调试功能;

(5) 单一电源(+9 V/0.2 A)供电,板内自带升压电路,产生编程电压 V_{PP} 以及+5 V 稳压器提供+5 V 电源;

(6) 用户程序灌写功能,读取芯片内部数据。

二、J1A 软件仿真器的结构

该仿真器电路很简单,尺寸很小,大小只有 91 mm×154 mm,其元件布置图如图 4-12 所示。其中,P1 为电源端口,S1 为电源开关,S2 为 V_{PP} 编程电压开关,P2 是与微机通信的端口,U5 为与用户目标系统相接的插口,U6 或 U7 是用于 J1A 片内 EPROM 编程时放置 J1A 单片机的插座(U7 适用于表面封装的芯片,U6 适用于 DIP 封装的芯片),S3 为复位按钮,U4 是板上仿真芯片,为另一种单片机 MC68HC705C9,负责完成与微机的通信以及 J1A 的仿真,J1~J17 为跳线设置,另外还有按钮 S4 以及用于显示演示程序执行效果的发光二极管 LED1,状态指示灯 LED2。

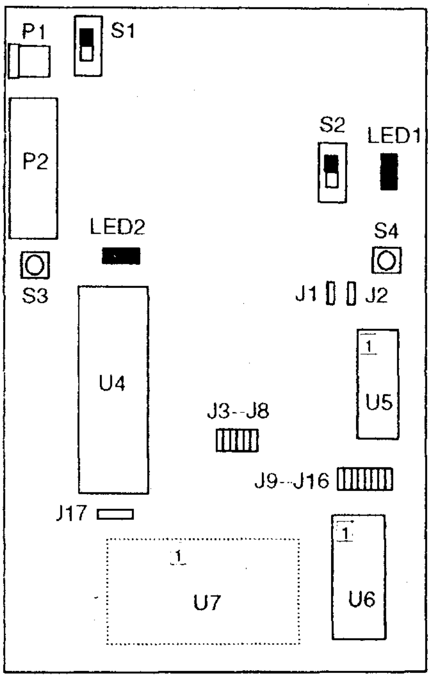


图 4-12 J1A 软件仿真器

这里要注意几个跳线的设置。J1,J2 用于演示程序的设置。该系统附带一个程序用于测试该仿真板,或用于演示仿真的效果。用户只要将该源程序汇编成 .S19 文件后,装载到板内执行,如果正常工作便可看到发光二极管 LED1 在闪动。J1 用以控制按钮 S4 是否有效。设置为有效后,当按下 S4 时,LED1 不会闪动;否则就会。J2 用以控制 LED1 是否允许工作,如果禁止则 LED1 在

演示程序执行时不会闪动。当用户调试自己的程序时，为了避免冲突，应将 J1，J2 都设置成无效状态，禁止 S4 与 LED1 工作。

J3～J8 用于设置 J1A 的 PB 端口片内下拉电阻是否有效。如短接就表示下拉电阻有效，否则就无效。

J9～J16 设置 PA 端口的片内下拉电阻。短接时，则 PA 端口内带下拉电阻。

J17 用于设置仿真器与微机的串行通信速率。设置为 1-2 短接时，表示波特率为 11.5 kbps；设置为 2-3 短接时，波特率为 19.2 kbps。

三、J1A 软件仿真器的软件系统

该种仿真器具有专用的调试软件 ICS05J1A.EXE，它可以安装在 Rapid 集成环境中由用户调用执行。用户亦可以在 DOS 环境下直接调用执行。在该调试环境里，用户可以做很多调试工作，正如 MMEVS 调试环境下一样，只是用户要注意它是非实时性的。如果用户不具有该软件仿真器的硬件系统，ICS05J1A.EXE 也可以独立运行，不需依赖仿真器。用户此时只能做离线调试，调试不依赖目标系统的软件部分。

1. ICS05J1A.EXE 的执行

在 DOS 提示符下键入

C)ICS05J1A [bw] [*filename*] *n* [/b19 200|/b115 200] [*]

其中，bw　　　表示以单色显示界面启动；

filename　　要装入的.S19 和.MAP 文件的名称；

n　　　所选择的串行通信口；

/b19 200　选择通信波特率为 19.2 kbps；

/b115 200　选择通信波特率为 11.5 kbps；

*　　　快速进入调试窗口，不显示版本信息。

2. ICS05J1A 的调试窗口

同 MMEVS 一样，该软件调试环境下也是多窗口显示形式，具有 CPU 窗口、源程序窗口、内存显示窗口、变量显示窗口和调试窗口等，如图 4-13 所示。

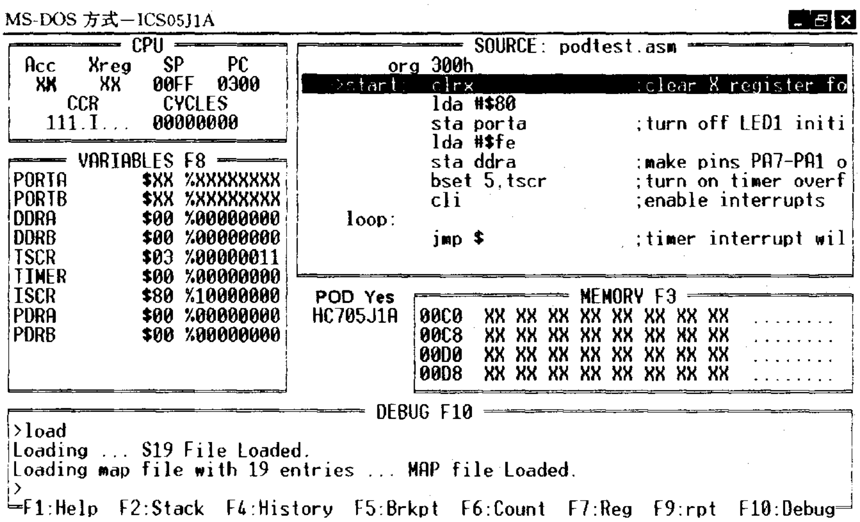


图 4-13　J1A 软件仿真器环境

这里要注意 CPU 窗口。在 CPU 窗口里，显示着累加器 A、寄存器 X、堆栈指针 SP、程序计数器 PC 以及状态寄存器 CCR 的值，另外还具有一个用以记录程序执行周期数的 CYCLES，用于确定时间。当需要了解某段程序执行的时间时，可以在该段程序启动前，先将该计数器清"0"，在该段程序的结束位置设置断点，然后执行程序直到遇到断点而结束。查看 CYCLES 的值便可知道程序的执行时间。最大计数值为 FFFFFFFFH，用户可以使用 CY 命令设置其计数初值，并计算在 2 MHz 内部时钟(晶振 4 MHz)情况下对应的时间值。

用户在调试窗口里使用调试命令来调试自己的程序。这些调试命令跟 MMEVS 系统的调试命令是兼容的，如表 4-9 所示。另外，还有一些热键的定义如表 4-10 所示。

表 4-9　ICS05J1A 命令

语　　法	意　　义
A *n*	设置累加器 A 的值
ACC *n*	设置累加器 A 的值
ASM *address*	修改语句
BELL [*n*]	产生鸣叫声
BF [.B|.W] *startrange endrange n*	在内存中填充一个数据
BR [*address* [*n*]]	设置断点
BREAKA [*n* [*address*]]	设置累加器 A 的断点
BREAKSP [*n*[*address*]]	设置堆栈指针的断点
BREAKX [*n*[*address*]]	设置变址寄存器 X 的断点
C 0|1	设置/清"0"进位标志
CAPTURE *address* [*address*...]	捕捉变化的数据
CAPTUREFILE [*filename* [R|A]]	打开捕捉数据存储文件
CCR *n*	设置条件码寄存器
CF[*filename* [R|A]]	打开捕捉数据存储文件
CLEARMAP	清除装入的.MAP 文件
CLEARSYMBOL	清除用户定义的符号
COLORS	设置显示器颜色
COUNTER [*address*]	设置指令执行计数器
CY [*n*]	设置周期数计数器
CYCLES [*n*]	设置周期数计数器
DASM [*address*|*startrange endrange*]	反汇编指定数据区
DDRA *n*	设置方向寄存器 A
DDRB *n*	设置方向寄存器 B
DUMP [.B|.W] *startrange endrange* [*n*]	显示指定内存数据
EVAL *n* [*op n*]	计算功能
EXIT	退出
G [*startaddr* [*endaddr*]]	开始快速执行程序
GO [*startaddr* [*endaddr*]]	开始快速执行程序
GOMACRO *filename*	遇到断点后执行指定文件中的命令序列

续表 4-9

语　　法	意　　义
GOTIL *endaddr*	执行程序到指定地址处停止
GOTOCYCLE *n*	执行程序到周期计数值达到指定值时停止
H 0|1	设置/清“0”H 标志
HELP [*topic*]	打开帮助窗口
HIDECHIP	关闭芯片图窗口
HISTORY	打开/关闭记录器
HISTORYLOG *n*	将记录器的指定内容保存在文件中
I 0|1	设置/清“0”I 标志
INFO	显示当前行有关信息
INPUTA *n*	设置 PA 端口的输入数据
INPUTB *n*	设置 PB 端口的输入数据
INPUTS	显示端口输入数据
INT [0|1]	设置 IRQ 状态
IRQ [0|1]	设置 IRQ 状态
LF [*filename*　[R|A]]	打开/关闭文件
LISTOFF	不显示每步运行结果
LESTON	显示每步运行结果
LOAD [*filename*]	装载.S19 文件
LOADMAP [*filename*]	装载.MAP 文件
LOADFILE [*filename*[*R*|*A*]]	打开/关闭文件
MACRO [*filename*]	执行指定文件中的命令序列
MACROEND	停止记录用户调试命令
MACROSTART *filename*	开始记录用户调试命令到指定文件
MD *address*	显示内存内容
MEM *address* [*n*...]	修改内存内容
MM *address* [*n*...]	修改内存内容
N 0|1	设置/清“0” N 标志
NOBR [*address*]	清除断点
NOCNT [*address*]	去掉指令计数器
PC *address*	设置 PC 指针
POD *n*	测试串行口
PORTA *n*	设置 PA 端口输出
PORTB *n*	设置 PB 端口输出
PROGRAM	灌写程序
QUIET	挂起窗口更新功能
QUIT	退出

续表 4-9

语　　法	意　　义
R	设置系统的寄存器
REG	显示寄存器的值
REM *text*	加入指示信息到批处理文件中
RESET	复位仿真器
RESET _ COUNTS	复位指令计数器
RESETGO	复位仿真器，然后执行用户程序
RUN [*startaddr* [*endaddr*]]	执行用户程序
SCRIPT [*filename*]	执行批处理文件中的命令
SHELL [*command*]	进入 DOS 环境
SHOW *address*	显示内存内容
SHOWCHIP	打开芯片显示图窗口
SHOWCODE *address*	显示代码
SHOWPC	显示 PC 指针所指处的代码
SNAPSHOT	保存现场数据到.LOG 文件中
SOURCE	源程序窗口显示方法设置
SP *n*	设置 SP 指针
SS [*n*]	执行 *n* 步
ST [*n*]	执行 *n* 步
STACK	显示堆栈区的内容
STEP [*n*]	执行 *n* 步
STEPFOR	连续执行程序
STEPTIL *address*	连续执行程序到指定位置
SYMBOL [*label value*]	定义一个符号变量
SYSINFO	显示系统信息
T [*n*]	单步执行
UPLOAD _ SREC *startrange endrange*	将指定内存的数据以 .S19 格式存入.LOG 文件
VAR [.B|.W|.S] *address* [*n*]	显示变量的数据
VER	显示版本信息
WAIT *n*	等待 *n* 个周期
WHEREIS *symbol*|*address*	显示指定符号的值
X *value*	设置 X 寄存器的值
Z 0|1	设置清“0”标志
ZOOM	改变显示窗口的大小

表 4-10　ICS05J1A 的热键

热　键	功　能	热　键	功　能
F1	打开帮助系统	PageUp	上移一页
F2	显示堆栈窗口	PageDown	下移一页
F3	进入内存窗口	Home	直接到达代码起点
F4	进入记录窗口	End	直接到达代码终点
F5	进入断点窗口	Alt-B	在光标处设置/清除断点
F6	进入指令计数窗口	Alt-C	在光标处设置/清除指令计数器
F7	显示寄存器文件窗口	Alt-F	查找字符串
F8	进入变量窗口	Alt-G	执行程序到指定位置
F9	重复执行上次命令	Alt-L	查找下一个
F10	进入调试窗口	Alt-P	设置 PC 到当前位置
↑	上移	Alt-Z	改变显示窗口形式
↓	下移	Shift-↑(or Alt-U)	重复输入前次命令
←	左移	Shift-↓(or Alt-D)	重复输入下次命令
→	右移	Esc	返回调试窗口

另外，通过命令可以显示一些窗口，例如，芯片各引脚图，在程序运行时，我们可以看到对应引脚高低电平的变化。还有堆栈窗口、指令计数窗口、断点窗口等。

从表 4-9 的调试命令表中，可以发现一些新的调试命令，这些调试命令只有在软件仿真器中才可以使用。这些命令的使用在一定程度上补偿了非实时性仿真的缺点。

断点的设置比 EVS 和 MMEVS 更加丰富，不仅可以在某条指令处设置，还可以用堆栈指针 SP、累加器 A、变址寄存器 X 来设置断点。例如，“BREAKA　55”可以指定当累加器 A 变化到指定值 55 时，立即停止程序执行；而“BREAKA　55　400”则表示当累加器 A 变化到指定值 55 时，在地址 400 处停止执行程序；BREAKA 不带参数时，表示取消该功能。

“BREAKSP　E0”表示当堆栈指针(SP)＝E0 时停止执行程序；而“BREAKSP　E0　400”则表示在(SP)＝E0 时，在地址 400 处停止执行程序；BREAKSP 不带参数表示取消该功能。

BREAKX 与 BREAKA 类似，条件只是针对变址寄存器 X。

想要跟踪记录数据的变化，使用调试命令 CAPTURE 就能做到。在快速、连续执行用户程序时，一旦指定的变量发生了变化，系统会自动保存当时的现场。用户可以捕捉到各种变化情形。

“CAPTURE PORTA”表示监视 PA 端口的变化。

“CAPURE D0 D1 D2”表示监视这三个单元的变化。

该命令要与 CAPTUREFILE 命令联合使用。CAPTUREFILE 用于打开记录存储文件，每次捕捉的现场数据都保存在打开的文件中。

对于指令计数器的设置，可以通过 COUNTER 命令在指定位置处设置计数器，记录程序执行经过该位置的次数。另外，还可以通过 CYCLES 或 CY 命令计算程序执行的周期数，从而得到程序运行的时间。

对于程序灌写过程，首先将 68HC705J1A 芯片正确放置在插座上，在保证装入的程序正确无误并且 MOR 寄存器(7F1H)数据已经设置的前提下，输入编程命令 PROGRAM，如果芯片未损坏，调试器工作正常，我们就可以在它的提示下进入到如图 4-14 所示的环境里。

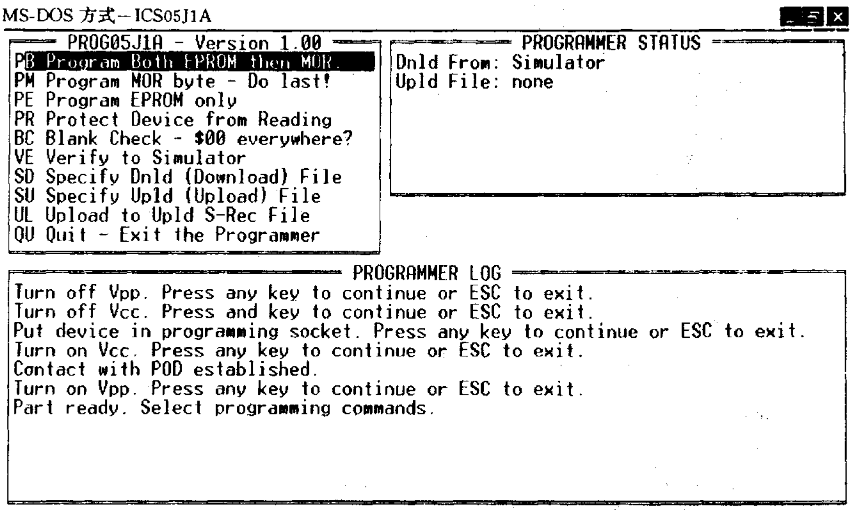


图 4-14 ICS05J1A 编程窗口

这时，用户可以根据自己的情况选择执行窗口中所列出的各种命令。

PB——对 EPROM 和 MOR 编程；

PM——对 MOR 寄存器编程(注意，应最后做)；

PE——对 EPROM 编程；

PR——禁止该芯片内容读出；

BC——检查该芯片是否为空，即检查各单元是否都为 00H；

VE——比较芯片内部 EPROM 的内容与仿真器的代码数据；

SD——确定下载的文件名；

SU——确定上载的文件名；

UL——将芯片内部数据保存到指定的上载文件中；

QU——退出编程状态。

在执行 PB 操作后，板上的状态灯会闪动，此时，用户耐心等待它停止闪动(1 分钟内)并观察微机屏幕上出现的信息。编程完毕后，将会自动执行校验命令，如果不正确会在屏幕上显示。

在取出芯片前，一定要先关 V_{PP}开关 S2，再关 V_{DD}开关 S1。在插入芯片后，上电顺序应是先接通 V_{DD}开关 S1，再接通 V_{PP}开关 S2。

4.4 MC68HC705J2 编程器

对 J2 系列的编程可以用非常简单实用的方法来实现。该方法操作快速，不依靠专门的软

件，也不依靠微机，特别适合灌写大量 OTP 型芯片。

在 MC68HC705J2 的内存中(0F01H～0FEFH)固化了一段程序，执行该段程序可以把外部 EPROM 芯片 2764 中的数据自动灌写到 68HC705J2 的内存中，从而实现自动编程。这种编程器的电路很简单，如图 4-15 中所示，用户完全可以自己组装出这个编程器。这个电路主要由三个集成电路组成，一个为待编程的 MC68HC705J2 芯片，一个为 12 位的计数器 MC14040B，还有一个是装有用户程序的普通 EPROM 芯片2764。

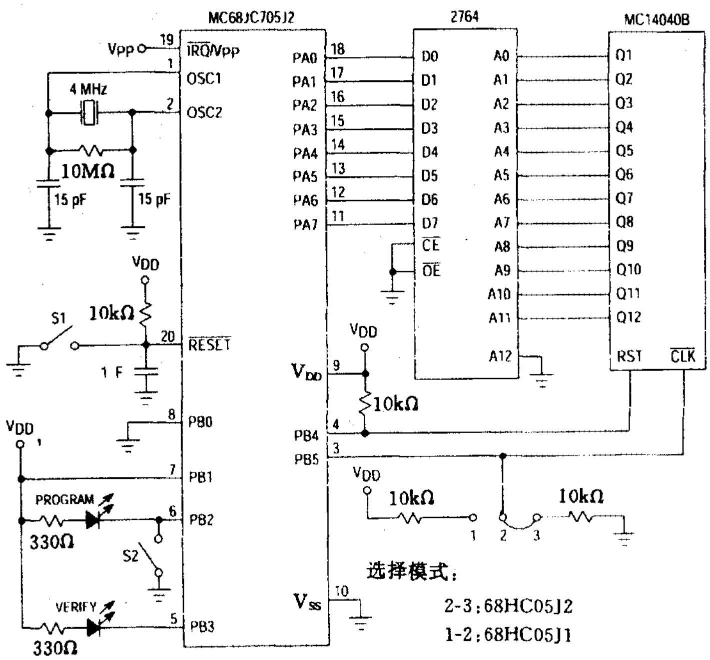


图 4-15　68HC705J2 编程电路

2764 内部的数据是用户事先灌写进去的。用户先将程序代码调试好，然后通过2764的编程工具灌写到该芯片内部。由于2764 EPROM 芯片是一个很常用的芯片，对该芯片的编程可以通过许多编程工具实现。用户存放在2764内部的数据其位置应与实际 MC68HC705J2 的用户 ROM 位置对应。对 J2 的编程，代码应放在2764的 700H～0EFFH 的区域里；用 J2 仿真 J1 芯片时，代码应放在2764的 300H～6FFH 的区域里。另外，还应包括 MOR 寄存器的数据，J2 的 MOR 地址在 F00H，J1 的 MOR 在 700H 处。

12 位计数器 MC14040B 用以产生2764的地址信息，其计数脉冲由 68HC705J2 的 PB5 产生，清“0”由 PB4 完成。

待编程的 68HC705J2 芯片在编程电压 $V_{PP}=16.5$ V 加在$\overline{IRQ}$引脚，且 PB1＝1、PB0＝0 时，内部结构发生变化而开始自装载程序过程，自动从2764中读取数据并写入到自己的 EPROM 空间中去。

68HC705J2 可以用来仿真 J1 单片机。由于 J1 与 J2 的内存分布有很大区别，因此，当用 705J2 来仿真 J1 时，一方面掩膜寄存器 MOR 要设置为 J1 仿真模式；另一方面在该电路中，PB5 应通过一电阻接＋5 V 或者通过一电阻接地。

电路中的两个发光二极管 LED，用以指示编程的选择与编程后校验结果。当 PB3 控制的

那个发光二极管在编程后不亮时，说明校验未通过，则需要检查编程电压 V_{PP}正确与否，或68HC705J2 芯片是否未用紫外线擦除原数据，再有就是芯片是否损坏。用户在操作时应注意前后次序，正确的步骤如下：

(1) 关掉电源，包括 V_{DD}与 V_{PP}；

(2) 把 2764 与 68HC705J2 分别放在对应插座上，注意方向别弄错；

(3) 选择 J2 的仿真模式：如要仿真 J1 则图中 1-2 短接，否则是 2-3 短接；

(4) 选择功能：打开开关 S2 表示编程后校验；闭合开关 S2 则表示仅作校验，比较单片机内部数据与2764内部的数据是否相等；

(5) 闭合开关 S1；

(6) 接通 V_{DD}电源；

(7) 接通 V_{PP}电源；

(8) 断开开关 S1；

(9) 等待编程灯熄灭和校验灯亮，如果检验灯很久不亮，则表示校验未通过；

(10) 关闭开关 S1；

(11) 断开 V_{PP}电源；

(12) 断开 V_{DD}电源；

(13) 取出单片机 MC68HC705J2，即告完成。

4.5 K 系列在线软件仿真器

K 系列也有一个简单实用的在线软件仿真器。由于 K 系列的用户程序量不大，用这种简单的调试工具便足够了。这种 K 系列在线软件调试工具一方面与目标系统相连，实现在线调试；另一方面又与微机的并行口(LPT1 或 LPT2)相连，在微机上实现多种调试功能，包括断点设置、事件捕捉以及编程功能等等。

一、K 系列在线软件仿真器的结构

这种仿真器电路也很简单，尺寸很小，大小只有 89 mm×81 mm。其元件布置图如图 4-16 所示，主要有电源端口、与微机的端口、目标系统的连接端口、下拉电阻的选择跳线以及有关编程电压的产生电路等。

P1 端口为电源输入端。在这里只需要接入+5 V 直流电源，而无需接入 V_{PP}编程电源(自带升压电路，产生 V_{PP})。

P2 端口为与微机的通信端口。这里要特别注意的是，该端口用于与微机的并行口(LPT)相连，实现微机与仿真器的数据交换。用户只要使用标准的 25 芯电缆线连接微机与仿真器即可。

P3 为目标系统与仿真器的端口。用户只要用 16 芯扁平线和转换接头把两者相连，程序执行的电平信号就可以传送到目标系统中去，同时，目标系统的状态信号也可以传送到仿真器中去，实现在线仿真。

插座 U2 为放置 68HC705K1 芯片的地方。该芯片内部 EPROM 未编程，可以通过这个仿真器把用户程序固化到其内部 EPROM 中。

发光二极管 LED1 按钮 S2 和跳线 J1，J2 用于演示程序。用户把系统所带的演示程序

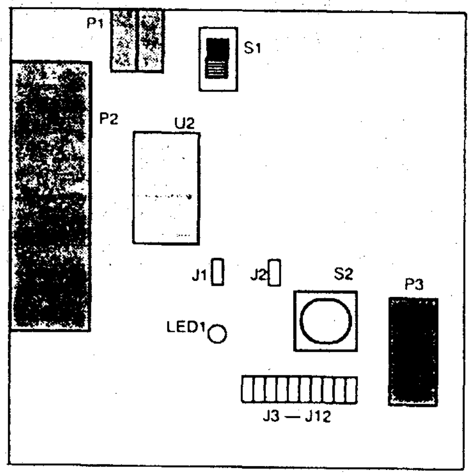


图 4-16　K 系列软件仿真器

PODTEST. ASM 汇编后，用 LOAD 命令装载到仿真器中执行，可以看到发光二极管 LED1 在闪动。此时按下 S2，LED1 会停止闪动。跳线 J1，J2 分别用于允许/禁止 LED1 和按钮 S2 的功能。在执行演示程序时，J1，J2 应短接；而在调试用户程序时，则应断开 J1，J2。

S1 为编程电压 $V_{PP}=16.5$ V 的开关。

J3～J12 跳线用于设置仿真 K 系列 PA 端口的下拉电阻。当这些跳线短接后，表示内部下拉电阻有效；否则无效。

二、仿真器的软件系统

这种仿真器的运行必须依赖于特定的调试软件，这里采用的是 ICS05K. EXE，另外系统还带有专用于 K 系列的汇编器 IASM05K. EXE。我们可以在该汇编器里进行编辑、汇编 K 系列程序，在 IASM05K 中也可以直接调用 ICS05K 的程序而进行调试工作。IASM05K 软件与 Rapid. EXE 软件功能是一样的，当然，我们可以用 Rapid. EXE 来进行编辑与汇编工作；而 IASM05K. EXE 只适合于 K 系列，其汇编器不适合其他系列。

1. ICS05K. EXE 的执行

该软件在 MS-DOS 操作系统下运行，也可以挂在 Rapid. EXE 上，由 Rapid 调用而执行。在 IASM05K 软件中，也可以由按功能键 F6 进入 ICS05K 的软件环境中。

ICS05K. EXE 可以在未与仿真器连接的情况下执行，但此时只能调试那些不依赖外部信号的部分程序。要做到在线调试，必须与仿真器配合使用。

ICS05K 的执行命令格式为

C〉ICS05K　[bw] [*filename*] [*n*]

其中，bw　　指定显示界面为黑白方式，缺省为彩色；

filename 需装载的程序名称；

n　　仿真器所使用的并行口号，2 为 LPT2；缺省为 1。

例如：

C〉ICS05K

C〉ICS05K　bw

C〉ICS05K　myprog

C〉ICS05K　myprog　2

2. ICS05K 的调试环境

当成功进入到 ICS05K 系统后，用户可以看到如图 4-17 所示的界面。在这里有多种窗口显示出来，反映出当时的调试状态。

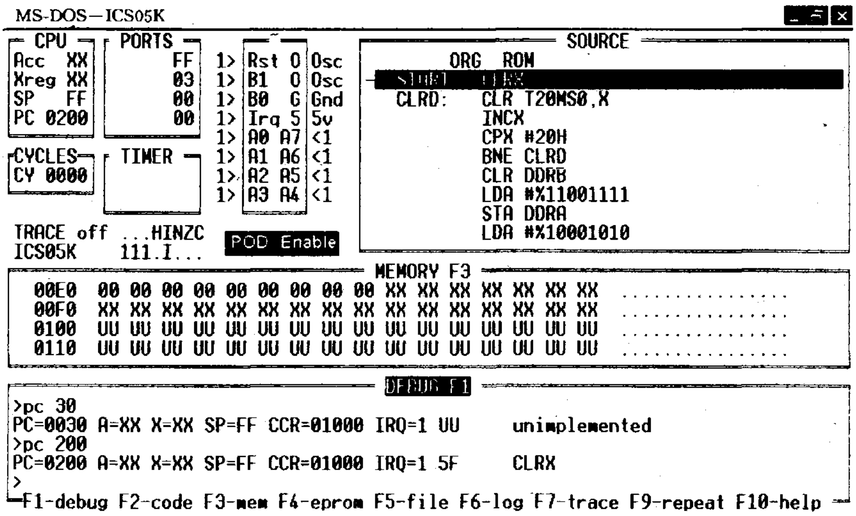


图 4-17　ICS05K 调试环境

CPU 窗口反映了累加器 A、变址寄存器 X、堆栈指针 SP 和程序计数器 PC 的内容变化，用户可以直接使用命令改变这些寄存器的值。例如，“A 30H”表示把 30H 赋给累加器 A。

周期计数窗口 CYCLES 显示出程序到目前已经历的周期数。这同上节中 ICS05J1A 仿真器的 CYCLES 计数器功能是一样的。由于该仿真器是非实时性仿真的，用户如要获得准确的时间值，就要依靠该周期计数器的数值来进行推算。在执行前先对其清零，然后执行程序，到达指定位置后观察 CY 的数值，CY 的值乘以一个周期的时间即为该段程序实际运行所用的时间。在该窗口里只显示了最低 2 位十六进制的计数值，实际上，该计数器为 4 位十六进制，最大计数值可以达 FFFFFFFFH。用户可以通过 CY 命令直接获得该计数器的十六进制和十进制的数值。

PORTS 窗口显示的是 PA 端口与 PB 端口的数据寄存器及方向寄存器的值。用户可以通过调试命令直接对各端口状态进行改变。例如，“DDRA　30H”表示把 PA 端口的方向寄存器值置为 30H。

定时器 TIMER 窗口显示的是内部定时器的状态，包括定时器的控制寄存器 TCSR 的设置以及计数器的计数值。

芯片引脚图形象地显示出各引脚的电平状态，这个状态与用户目标系统是对应的。目标系统在各引脚上的电平变化都可在该图中表现出来。

CODE 窗口为程序显示窗口。在窗口里，我们可以看到程序执行的位置。窗口里显示的程序可以是源程序形式，用户定义的变量、注释都可直接显示出来。前提是已装入用户程序的 .MAP文件，否则只能以反汇编的形式表示出来。

内存（MEMORY）窗口显示的是内存的数据。用户可以使用 MD 命令显示指定的内存单元。窗口中 XX 表示该内存单元的数值是未初始化过的。

调试窗口(DEBUG)为用户调试命令输入的地方，同时也是显示每步调试结果的地方。

另外还有一些二级窗口，这些窗口只有在用户输入相应命令时才被显示。例如，文件显示窗口，LOG 文件窗口，Personality EPROM 窗口，帮助文件窗口等。

文件显示窗口可以通过功能键 F5 激活而进入。该窗口用于显示用户在调试过程中经常需要查看的文件。

LOG 文件窗口可以通过功能键 F6 激活而进入。该窗口用于显示用户在调试中所做的记录文件。

在 68HC705K1 的内部有一个称为 Personality EPROM 的存储区，用户可以用功能键 F4 来激活并显示该部分的内容，显示方式如图 4-18 所示。进入到该窗口后，用户可以用空格键修

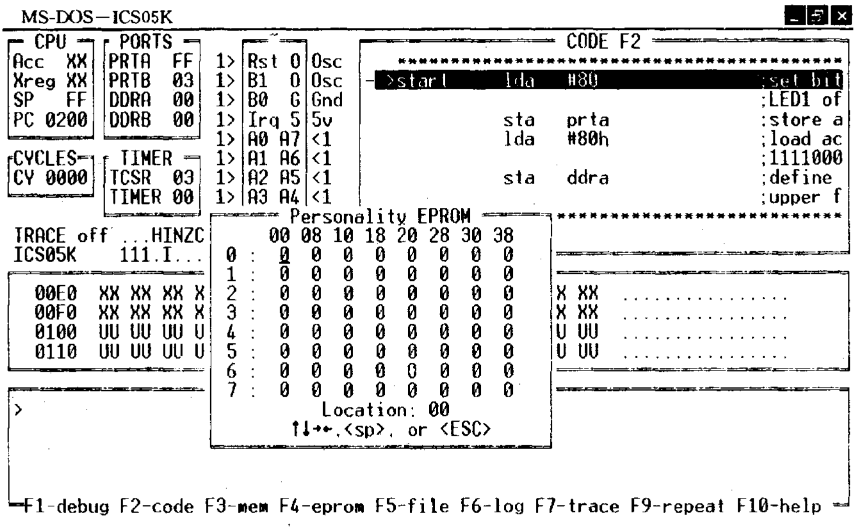


图 4-18 Personality EPROM 窗口

改其内部的数据。当退出该窗口时，系统会自动将窗口内部的数据保存在 EPROM.PER 文件中。每当进入仿真系统时，系统会自动在当前目录下寻找该文件，将其数据设置在该 EPROM 区里。用户要注意，MOR 寄存器的数据是包括在该 EPROM 区中的，因此，用户要根据自己的情况设置好，并且要在灌写程序之前设置好。

3. ICS05K 的调试命令

ICS05K 的调试命令与 ICS05J1A 的调试命令类似，也具有捕捉命令、丰富的断点设置命令等，详见表 4-11。还有一些热键的定义详见表 4-12。

4. MC68HC705K 内部 EPROM 编程步骤

首先，在芯片未放到插座 U2 上的情况下，接通开关 S1，检查编程电压 V_{PP}的值是否正确，正确值应为 16.5 V。

然后关断 S1，关断＋5 V 电源，将芯片插入到 U2 上。

表 4-11 ICS05K 调试命令

命令	解释
ASM [*add*]	修改程序
BF *add*1 *add*2 *n*	填充数据
BR [*add*]	设置/清除断点
BREAKA [*n*]	设置累加器 A 断点
BREAKSP [*n*]	设置堆栈指针 SP 断点
BREAKX [*n*]	设置寄存器 X 断点
CAPTURE [*filename*]	允许 I/O 捕捉功能
CAPTUREOFF	禁止 I/O 捕捉功能
CLEARMAP	清除 .MAP 文件
CY [*n*]	设置周期计数器初值
EXIT	退出 ICS05K
G	快速执行程序
GOTIL *add*	执行程序到指定位置
HELP	显示帮助信息
INPUTA *n*	设置仿真 PA 输入数据
INPUTB *n*	设置仿真 PB 输入数据
INPUTS	显示仿真输入的数据
IRQ *n*	设置仿真 IRQ 的电平
LOAD	装载 .S19 文件
LOADH	装载 .HEY 文件
LOADMAP	装载 .MAP 文件
MD *add*	显示指定内存的数据
MM *add* [*n*]	修改内存内容
NOBR	取消断点
PROGRAM	固化用户程序
POD *n*	重置仿真器与微机通信的端口号
SCRIPT *filename*	执行批处理文件
RESET	复位系统
SOURCE	改变程序显示方式
ST [*n*]	单步执行程序
STEPFOR	连续单步执行直到断点处
STEPTIL *addr*	连续单步执行直到指定位置处
SYMBOL *sym val*	定义某变量
UPLOAD	将仿真器的代码数据以 .S19 形式保存
WHEREIS *add*	显示指定符号的值

表 4-12 ICS05K 系统的热键

热键	功能	热键	功能
F1	进入调试窗口	F9	重复上次命令
F2	进入程序窗口	F10	激活帮助文件
F3	进入内存窗口	Alt-B	在指定处设置断点
F4	打开个人 EPROM 窗口	Alt-F	查找指定字符
F5	打开显示文件窗口	Alt-G	执行程序到当前光标处
F6	打开 LOG 文件窗口	Alt-L	重复上次查找的字符
F7		Alt-P	设置 PC 到当前光标处

在调试窗口中输入 PROGRAM 命令，此时，系统会有一系列的操作提示，请按提示做即可。进入编程窗口，如图 4-19 所示。

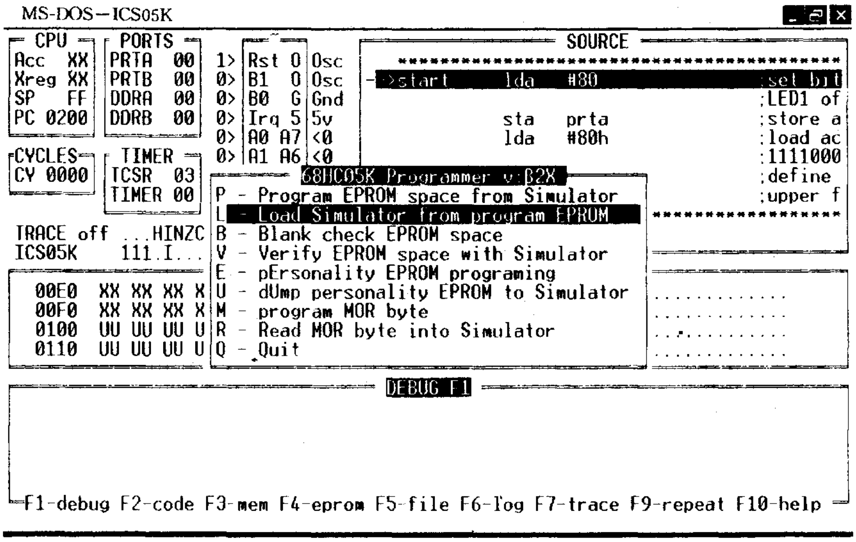


图 4-19　ICS05K 编程窗口

这时，用户可以根据自己的情况选择执行各种命令。

P——对 EPROM 编程；

L——装载芯片内部数据到仿真器中；

V——验证芯片内部数据与仿真器数据的一致性；

E——个人 EPROM 编程；

U——装载芯片内部的个人 EPROM 到仿真器中；

M——对 MOR 编程；

R——读 MOR 内容到仿真器；

Q——退出。

编程后，按照系统提示先关掉 S1 开关，再关掉＋5 V 电源，然后取出芯片。

第五章　MC68HC05J 系列的应用

5.1　键盘矩阵端口设计

在本节中，我们用 J 系列的 MC68HC05J1A 单片机构成一个键盘矩阵译码系统。从这个例子中我们可以学会如何使用键盘中断功能，如何使用低功耗工作方式 STOP 以及如何扫描键盘矩阵等。该例给出了详细电路以及全部控制主程序与两个子程序。

一、设计思想

这里需要读一个 4×4 矩阵按键，共 16 个按键。为了读取 16 个按键的状态，必须采用动态扫描的方法，否则就无法用 J 系列单片机实现。采用动态扫描的方法只需要 8 条 I/O 线就可实现，其中 4 条作为扫描线，4 条为读状态线。为此，可以使用 J1A 的 PA0～PA3 作为读状态线，这 4 条 I/O 线可以具有中断功能，且内部也具有下拉电阻，可方便地实现读键盘矩阵。由于 PA 端口具有中断功能，我们就可以使用该种中断来唤醒 STOP 的状态。STOP 状态的使用可以减少系统处于待机状态时的功耗，这在使用电池供电的系统中尤为重要。扫描信号线采用 PB0～PB3，在待机状态时，PB0～PB3 全部输出高电平。此时，当任一按键按下时，PA0～PA3 中的某条线就会得到一个上跳信号，从而引起中断产生，并使系统退出 STOP 状态。退出后才重新扫描，确认是哪一个键按下。确认后把键号用 4 位二进制数来表示，送到 PA4～PA7 中去，用发光二极管 LED 来显示扫描结果。PA4～PA7 引脚的驱动能力强，可以直接驱动 LED，而无须三极管驱动。

根据以上设计思想设计的电路如图 5-1 所示。图中 J1A 的振荡元件采用三脚陶瓷滤波器，在外部复位电路中采用了一个低电压检测元件 MC34064。该元件为小三极管般大小，常用在复位电路中，以提供低电压复位功能。当电源电压 V_{DD} 降至一定程度时，它会马上强行使 $\overline{\text{RESET}}$ 引脚为低电平，保证单片机的正常复位。

二、软件设计

控制程序主要由主程序部分、IRQ 外部中断服务子程序、编码子程序以及延时子程序组成，各子程序的流程图如图 5-2，5-3，5-4，5-5 所示，其他程序很简单。初始化各 I/O 端口与中断系统后便进入低功耗状态，等待按键中断产生；当有键按下时，便引起 IRQ 中断。IRQ 中断服务程序负责扫描键盘，确认是哪一个按键，确认后，计算其编码值，并送到 PA4～PA7 端口通过 4 个发光二极管显示。

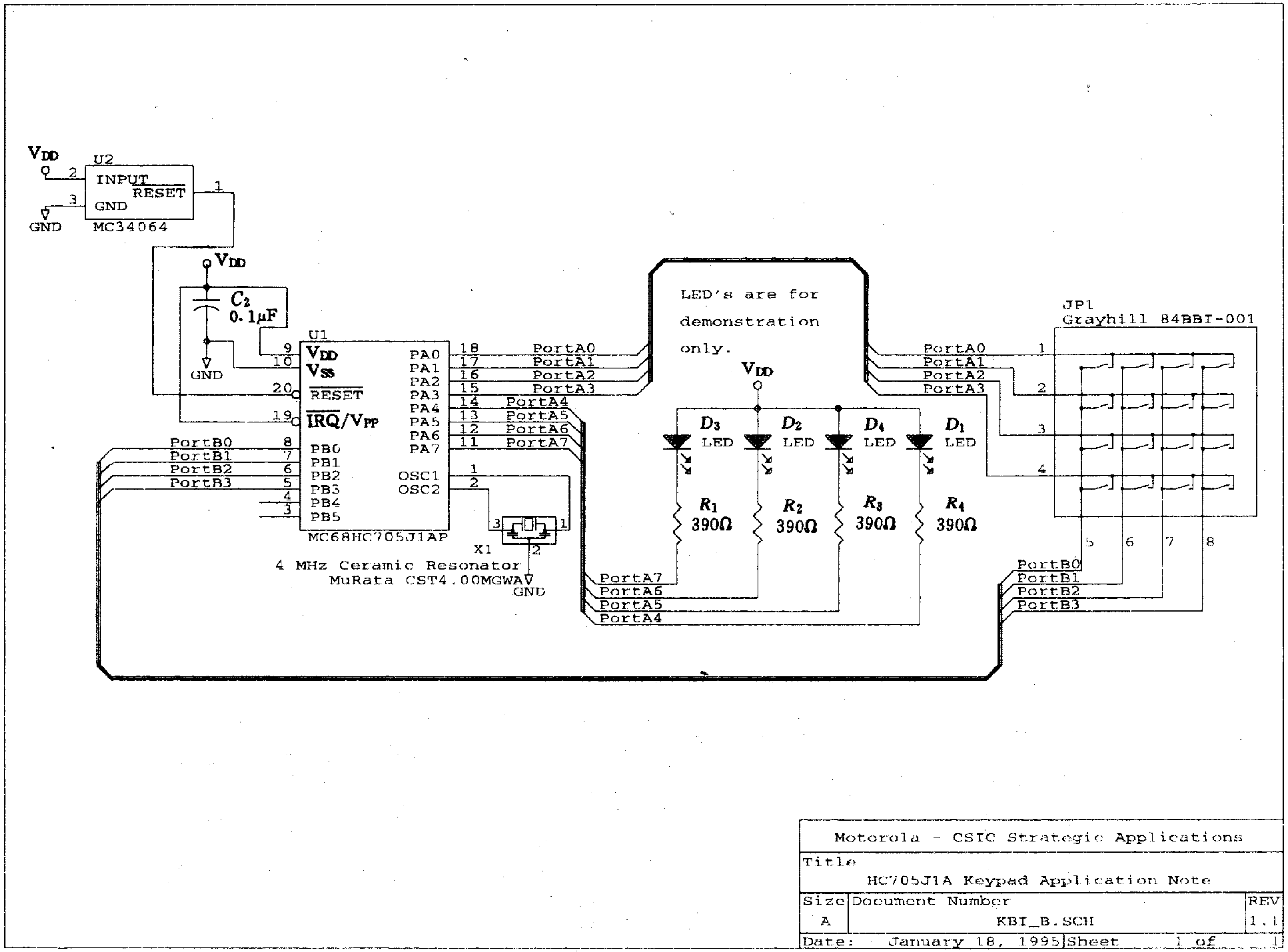


图 5-1 键盘矩阵译码系统电路图

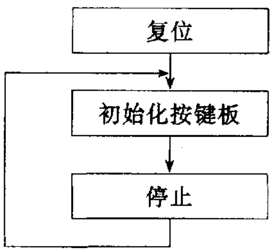


图5-2　主控程序流程

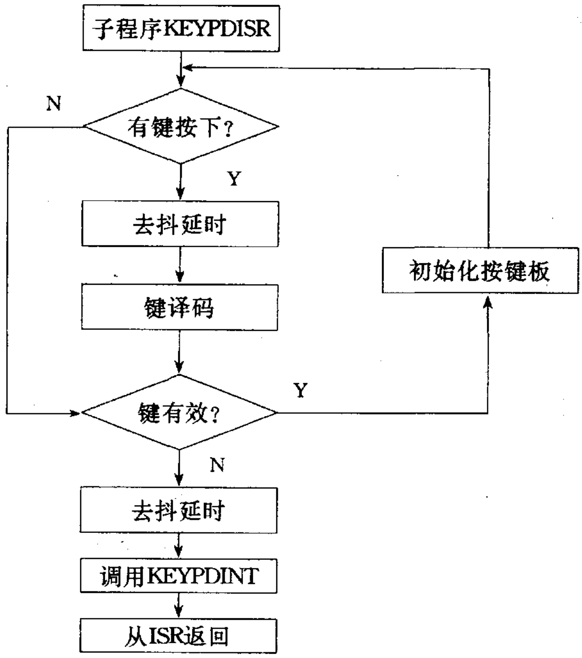


图5-3　外部中断子程序流程

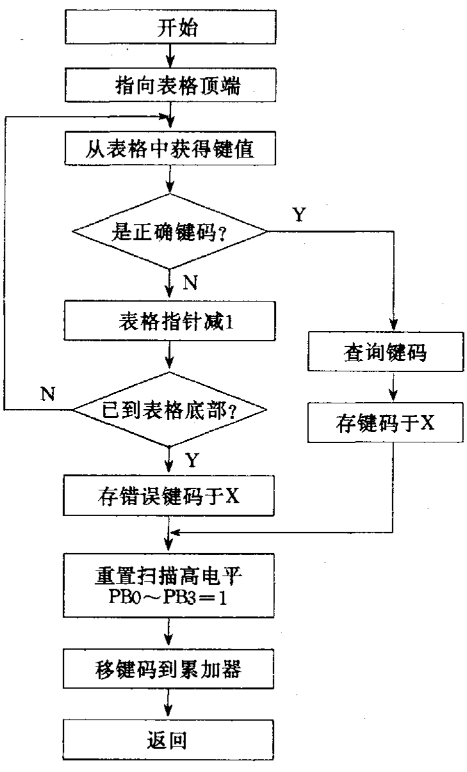


图5-4　编码子程序流程图

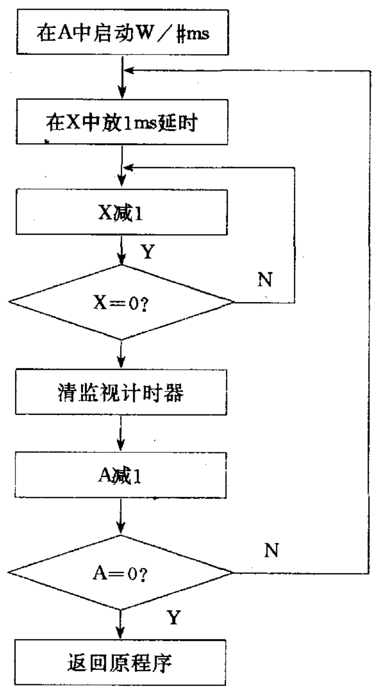


图5-5　延时子程序流程图

三、程序清单

```
                  org     MOR
                  db      PIRQ. +OSCRES.      ;Enable Port A Interrupts
                                              ;If used on a mask Rom part,
                                              ;be sure to specify this option.
***************************************************************************
*                         Program Initialization                          *
* This routine sets up the high nibble of port  a to drive LED'S          *
* with it's high sink current. Due to the use of sink current,            *
* the LED's will be on when an low is output and off when a high is output. *
* It then calls the keypad _Init routine to setup the ports to            *
* interrupt the processor when a key is pressed.                          *
* To prevent floating inputs and associated high current draw.            *
* the HC705J1A has pulldown devices on all I/O pins. This                 *
* initialization should enable these pulldowns on unused I/O              *
* pins. RESET _enables the pulldowns, so no code is required.             *
***************************************************************************
                  org     EPROM
Start:
KeyPdInt _Init:
                  lda     #$F0                ;Set the high nibble as output high.
                  sta     PORTA               ;This enables output drive
                  STA     DDRA                ;for LED's but turns them off.
                  jsr     KeyPad _Init        ;Set up the ports to interrupt
                                              ;on a keypress.
***************************************************************************
*                         KeyPdInt Main Program Loop                      *
* This section simply services the COP watchdog and then enters STOP mode. *
* All other program execution is contained in the KeyPdInt _Isr,the external *
* interrupt service routine for this code.                                *
***************************************************************************
KeyPdInt _Body:
                  STOP                        ;Execute STOP instruction to put
                                              ;MCU in lowest power mode.
                                              ;The keypad can exit from STOP.
                                              ;STOP clears the I bit so CLI is
                                              ;not needed.
                                              ;When RTI returns from ISR, I bit
                                              ;will be clear, enabling ints.
                          bra KeyPdInt _Body  ;Infinite loop to stay in STOP.
***************************************************************************
*                         IRQ Interrupt Service Routine                   *
* This is the external interrupt service routine. Both the external       *
```

```
* interrupt pin IRQ  and the keypad interrupts use this routine. the real         *
* work of the program is done within this service routine.                       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
KeyPdInt _ Isr:     ;Any decoding of external interrupts should be done here.
                    ;The external and Keypad interrupt share this vector.
keyPdInt _ Isr010:
                    jsr     KeyPad _ Body         ;See if a key is pressed
                                                  ;If no key down, return
                    beq     KeyPdInt _ Isr090
                    lda     # $ 4                 ;Debounce key for 4 ms
                    jsr     DelaymS2 _ Body       ;Jump to delay routine
                    jsr     KeyPad _ Body         ;Get the keypress
KeyPdInt _ Isr020:
                    beq     KeyPdInt _ Isr090     ;If no key down, return
                    ;Operations that are to be performed based on a key should
                    ;be placed here. This example will just flash the code.
                    coma                          ;Complement the result
                                                  ; because the LED's are
                                                  ; negative logic.
                    lsla                          ;Move the 4bit result into
                    lsla                          ; the high nibble.
                    lsla
                    lsla
                    sta     PORTA                 ;Output the result for view.
                    lda     # ! 200               ;Show the result for 200 ms.
                    jsr     DelaymS2 _ Body
                    lda     # $F0                 ;Turn off the LED's
                    sta     PORTA
KeyPdInt _ Isr080:
                    bra     KeyPdInt _ Isr010     ;Back to beginning to repeat
KeyPdInt _ Isr090   lda     # ! 10                ;Delay 10 ms
                    jsr     DelaymS2 _ Body       ;Debounce the release
                    jsr     KeyPad _ Init         ;Set up the port to interrupt
                    bset    IRQR,ISCR             ;Clear any interrupt requests
                                                  ;generated due to key bounce
                    rti                           ;Return form Interrupt.
                                                  ;Interrups can happen in any
                                                  ;code in the main routine after
                                                  ;this ISR has been called once.
                                                  ;Remember this when changing
                                                  ;the main routine!
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                         Keypad Initialization                                   *
```

```
* This code sets up the low nibble of ports A and B to decode a 4×4 matrix       *
* keypad. This does not affect the high nibble of the port data or data          *
* direction registers.                                                           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
KeyPad _ Init:
            lda     ddra                    ;Set the low nibble of port A as input
            and     #$F0                    ; without affecting the high nibble.
            sta     ddra                    ;This also enables the pulldowns.
            lda     portb                   ;Set the low nibble of port B to high.
            ora     # $ 0F                  ;This will defeat the pulldowns on
            sta     portb                   ; port A if a key is pressed.
            lda     ddrb                    ;Set the low nibble of Portb as output.
            ora     # $ 0F
            sta     ddrb
            clr     PDRA                    ;Ensure that the pulldowns on port A
                                            ; are not disabled.
            rts                             ;Return to calling program.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* KeyPad _ Body                                                                  *
* This subroutine decodes a 4×4 matrix keypad on port B.                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
KeyPad _ Body:                              ;Load X with the offset of the last
                                            ;entry in the table
            ldx     #{KeyPad _ Table _ Top-KeyPad _ Table}
KeyPad010:
            lda     portb                   ;Get value in port B
            and     # $F0                   ;Do not allow high nibble to change
            ora     keyPad _ Table+1,x      ;Get key decode value from table
            sta     portb                   ;Write to port
            lda     porta                   ;Get value in port A
            and     # $ 0F                  ;Throw out columns to read only rows
            cmp     KeyPad _ Table,x        ;See if high nibble bit was pulled low
            beq     KeyPad030               ;If key found, branch
            decx                            ;Decrement X three times to point to
            decx                            ;  next value in table
            decx
            bpl     KeyPad010               ;If not below bottom of table
                                            ;  try again.
            ldx     # $ 00                  ;A key was not decoded, so:
            bra     KeyPad035               ;Return with null character
KeyPad030:
            lda     KeyPad _ Table + 2,     ;Load key code into ACC.
                    x
            tax                             ;Store in X for now.
```

```
KeyPad035:          lda     portb               ;'Help' the pulldowns by driving the
                    and     #$F0                ;  lines low. This minimizes current
                    sta     portb               ;  draw while debouncing.
                    txa                         ;Get result back to ACC.
                    tsta                        ;Set the flags so calling routine
                                                ;  can use them for decisions.
KeyPad040           rts                         ;Return with result value in ACC
                    ;Table of keypad decode values and codes.
                    ;Fill in your own key codes. Codes must be 1
                    ;  byte each. Currently limited to 4 bits to
                    ;  display on PA[4..7].
                                                ;Row  Column
KeyPad_Table        DB      $01,$01,$1          ;PA0  PB0
                    DB      $01,$02,$2          ;PA0  PB1
                    DB      $01,$04,$3          ;PA0  PB2
                    DB      $01,$08,$A          ;PA0  PB3
                    DB      $02,$01,$4          ;PA1  PB0
                    DB      $02,$02,$5          ;PA1  PB1
                    DB      $02,$04,$6          ;PA1  PB2
                    DB      $02,$08,$B          ;PA1  PB3
                    DB      $04,$01,$7          ;PA2  PB0
                    DB      $04,$02,$8          ;PA2  PB1
                    DB      $04,$04,$9          ;PA2  PB2
                    DB      $04,$08,$C          ;PA2  PB3
                    DB      $08,$01,$F          ;PA3  PB0
                    DB      $08,$02,$E          ;PA3  PB1
                    DB      $08,$04,$F          ;PA3  PB2
KeyPad_Table_       DB      $08,$08,$D          ;PA3  PB3
Top
****************************************************************************
* Delay for Xms                                                            *
* Inner loop delays 1 ms. Outer loop counts ms.                            *
* Number of ms in passed through the accumulator.                          *
****************************************************************************
DelaymS2_Body:                                  ;JSR EXT to get here     6
DelaymS2010         ldx     #$F8                ;Load delay into X       2 ──┐
delaymS2020         decx                        ;  Decrement delay       3┐  │
                    nop                         ;  burn 2 bus cycles     2│  │
                    bne     DelaymS2020         ;  Branch if not done    3┘  │
                    stx     COPR                ;Service the WDOG        5   │
                                                ;Note that X will            │
                                                ;always be zero here     3   │
                    brn     *                   ;Burn 3 bus cycles       3   │
                    deca                        ;decrement # of ms       3 ──┘
                    bne     DelaymS2010         ;branch if not done
```

```
Delayms2030         rts                      ;return         6
**************************************************************
*            Interrupt and Reset vectors for Main Routine       *
**************************************************************
                    org      RESET
                    fdb      Start
                    org      IRQ_INT
                    fdb      KeyPdInt_Isr
```

5.2 J系列单片机与 RS232 通信

这一节将介绍J系列单片机与RS232端口实现串行通信的一种方案。由于J系列内部不具备串行通信模块，而我们又常常要与一些具有 RS232 串行端口的设备进行数据交换，为此，我们提供一种用程序控制仿真串行口模块读取和发送串行数据的方法，读者可以参照应用在类似的程序设计中，实现半双工串行通信。

一、设计思想

串行通信时，数据是以串行的形式发送出去的。它由起始位、数据位、奇偶检验位以及停止位组成。在本方案中采用的是 NRZ 格式，即 1 个起始位、8 个数据位、1 个停止位，如图5-6所示。

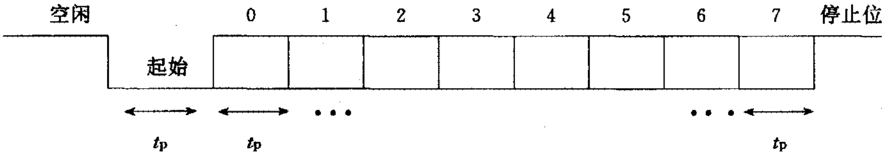


图5-6　串行通信传输数据的格式

由于J系列无串行通信端口，这就要求程序控制一位一位地发送与接收数据，用软件来实现硬件的功能。接收时，当接收到起始位以后，单片机就要根据波特率的大小，每隔一段时间就去读传送过来的电平信号。为了准确获取数据、排除噪声干扰，对每位数据都要连续读三次，三次不同就说明有噪声。读停止位时，如为低电平(停止位只能为高电平)，则此时接收的数据是不合格的。各位数据接收进来后，再组成一个 8 位二进制数。发送数据时，先发一个低电平作为起始位，然后再从数据的低位开始发送各位，最后发送一个高电平作为结束。如不再发送数据，应保持输出为高电平。

根据以上设计思想设计的电路如图 5-7 所示。这里使用 PB0,PB1 分别作为接收与发送数据的输入/输出口。由于 RS232 的信号电平为 ±12 V，因此使用电平转换芯片 MC145407，将 0 V～5 V 的信号转换为±12 V 信号输出；同时，也将外界传送进来的±12 V 信号转换为 0 V～5 V信号输入到单片机。该种电平转换芯片无需接＋15 V 电源，其内部自带升压电路。图中，单片机是采用的 J1A 芯片，当然也可以使用其他种类芯片，这里只用了 PB0,PB1 与 PA4，PA5 资源。PA4，PA5 用以显示通信接收过程中是否出现错误，分别指示有噪声以及结构错信息。用户当然可以根据自己的情况不使用这一功能。

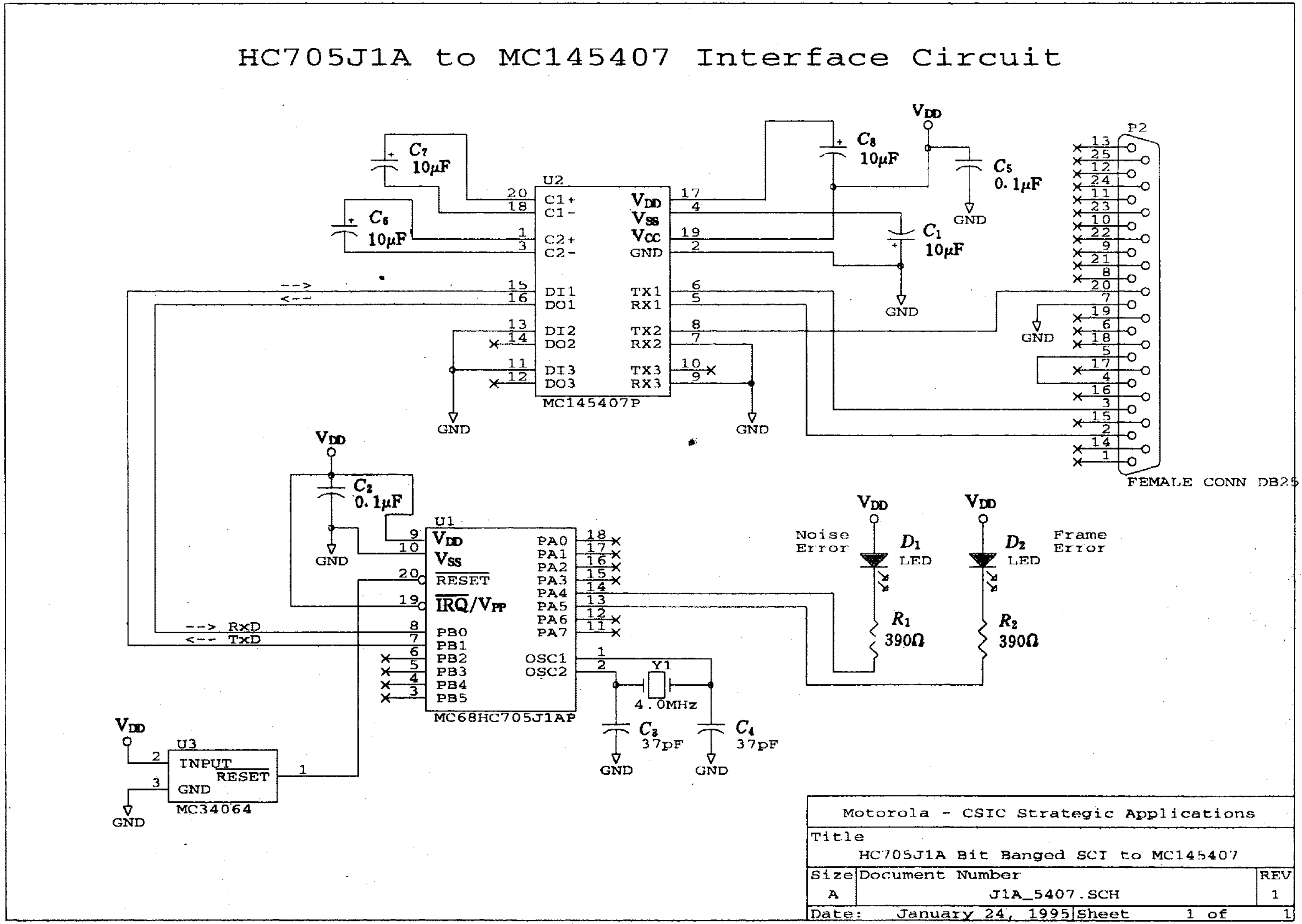


图 5-7　HC705J1A 与 RS232 通信电路图

二、软件设计

软件主要包括初始化程序 INIT，发送程序 PUT _ CHAR，接收一字节程序 GET _ CHAR，接收一数据位程序 GET _ BIT 以及延时程序 DELAY _ 13A。

PUT _ CHAR 完成发送操作，把一字节的数据一位位地发送出去。该程序流程图如图 5-8 所示。

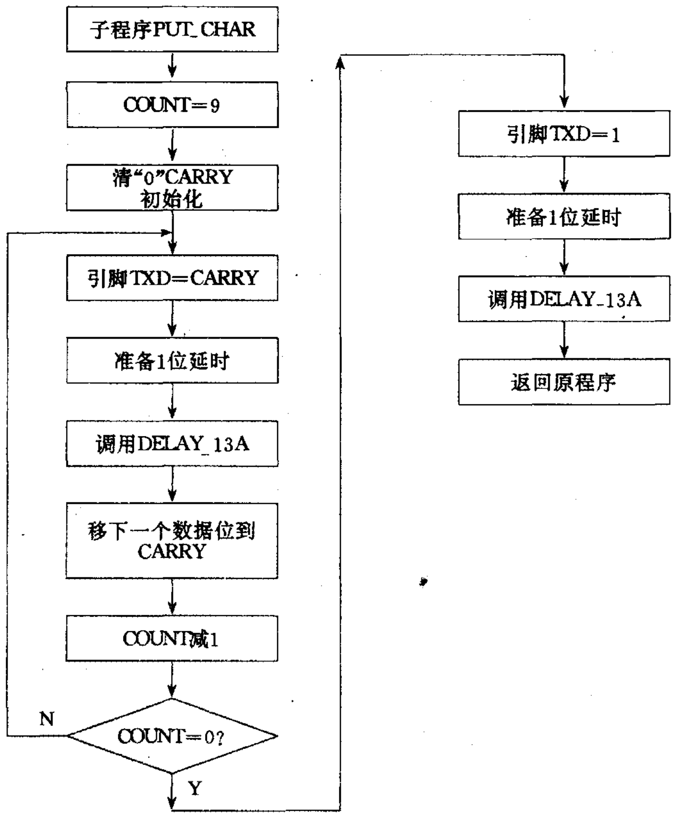


图5-8　数据发送子程序流程图

GET _ CHAR 完成接收一字节操作。程序通过调用 GET _ BIT 接收 8 位数据组成一字节数据。GET _ CHAR 还负责把噪声错误和结构错误通过发光二极管显示出来。其流程图如图 5-9 所示。

GET _ BIT 子程序仅仅完成对输入引脚 PB0 的三次采样。该程序使用指令 BRSET 来实现采样，每次采样的结果保存在进位标志中。其流程如图 5-10 所示。

延时子程序 DELAY _ 13A 用于波特率的设置，决定每位之间的时间间隔。该程序延时时间为 13×ACC+12 个 CPU 时钟，ACC 为累加器的数值。流程如图 5-11 所示。

初始化程序 INIT 仅在开始时执行一次，用以初始化 I/O 端口。流程图如图 5-12 所示。

整个程序流程如图 5-13 所示，它不停地接收数据并把接收的数据又发送出去。

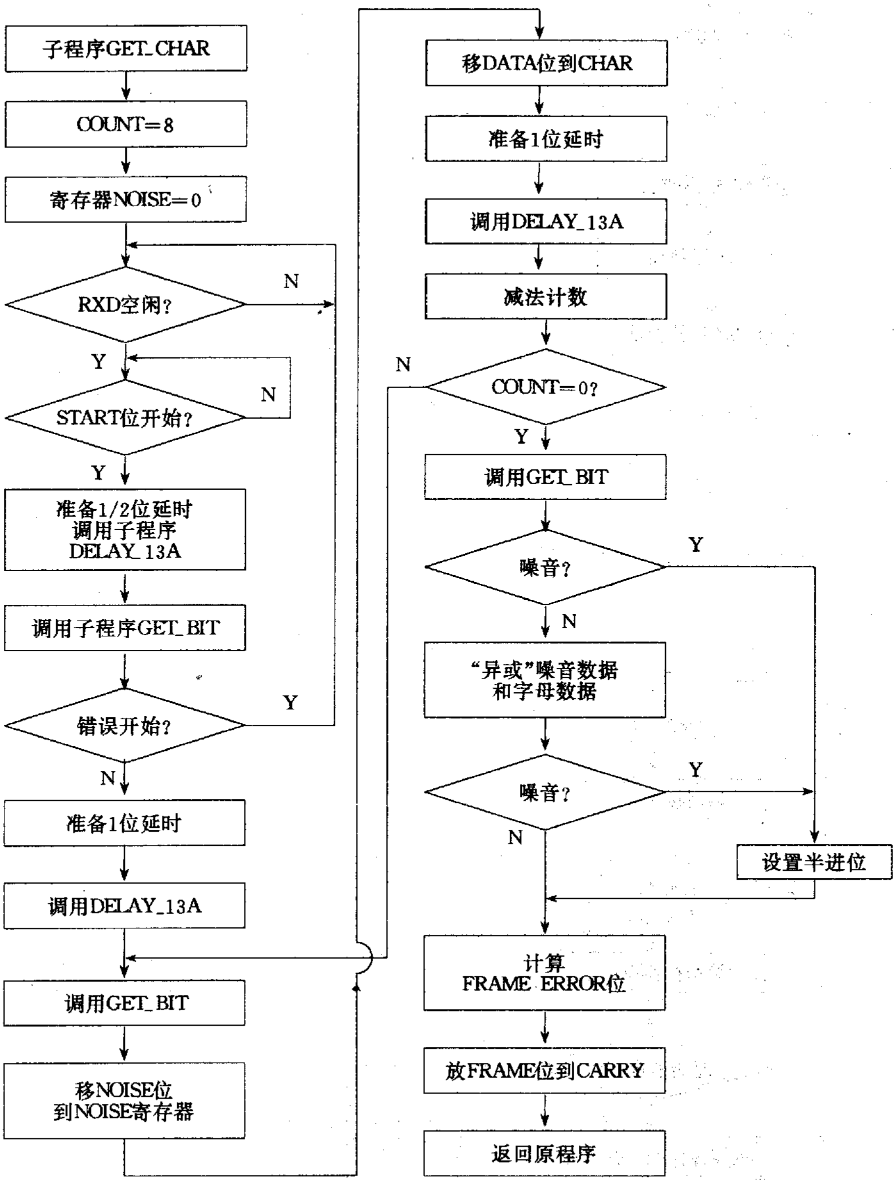


图5-9　接收一字节数据的子程序流程

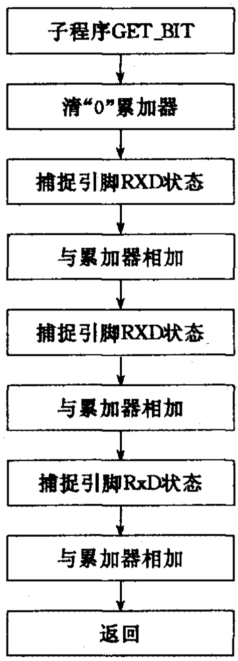


图5-10　接收一位数据的子程序流程

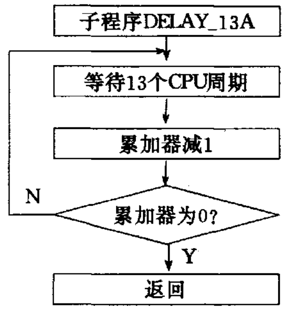


图5-11　延时子程序流程图

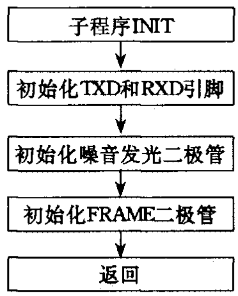


图5-12　初始化子程序流程图

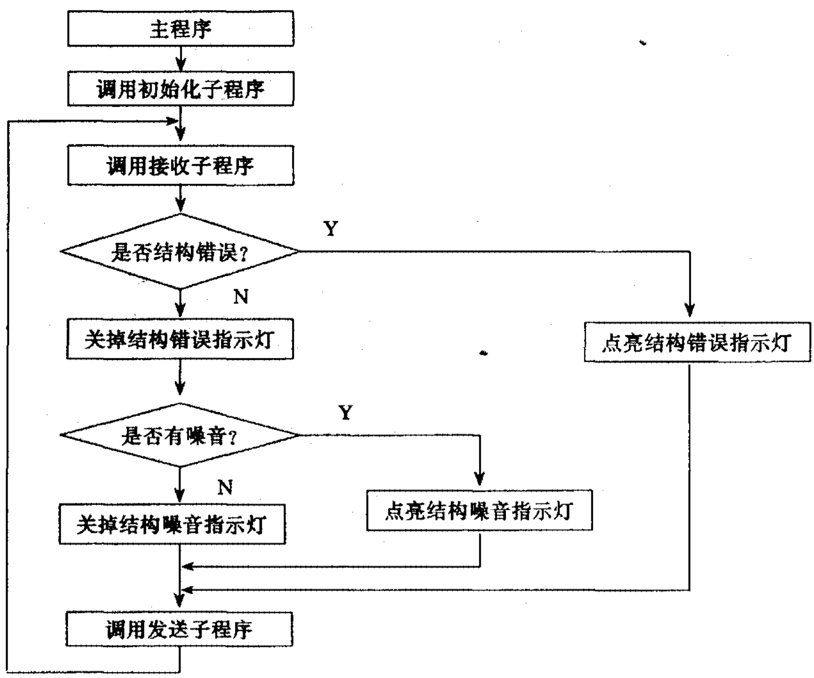


图5-13　总的程序流程图

三、程序清单

```
#INCLUDE 'H705J1A.FRK'                       ;Include the equates for the
                                             ;HC705J1A so all labels can
                                             ;be found.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                MOR Bytes Definitions for the Main Routine                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   org      MOR
                   fcb      $20
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                  Equates and RAM Storage                                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * *I/O Pin Equates:
serial_port        equu     $01              ;port used for serial port pins
status_port        equ      $00              ;port used for driving LED's.
noise              equ      4                ;pin # for noise LED
frame              equ      5                ;pin # for frame LED
rxd                equ      0                ;pin # for receive data pin
txd                equ      1                ;pin # for transmit data pin
* * *Program Constant Equates:               ;Baud rate select table:
BAUD_SEL           equ      $08              ;BAUD_SEL  4MHz osc    2MHz osc
                                             ;  $04     19.2k bps   9 600 bps
                                             ;  $08     9 600 bps   4 800 bps
                                             ;  $10     4 800 bps   2 400 bps
                                             ;  $20     2 400 bps   1 200 bps
                                             ;  $40     1 200 bps     600 bps
                                             ;  $80       600 bps     300 bps
* * *RAM variable allocation:
                   org      RAM
char               rmb      1                ;data register for sci
count              rmb      1                ;temp storage variable
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*main——example program that continually echoes back received characters.        *
*input cond.——reset                                                             *
*output cond.——none (infinite loop)                                             *
*stack used——4 bytes                                                            *
*varlables used——none                                                           *
*ROM used——28 bytes                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              org      ROM                   ;start at the top of ROM
main          rsp                            ;reset the stack pointer
              jsr      init                  ;initialize port pins
main_loop     jsr      get_char              ;receive one byte of data from rxd pin
```

```
                bcc     no _ frame _ error          ;branch if no noise occured
                bclr    frame,status _ port         ;turn on frame LED
                bra     continue                    ;don't check for noise--
                                                    ;it's undefined
no _ frame _ err  bset    frame,status _ port         ;turn off frame LED
                bhcs    noise _ error               ;branch if noise occured
                bset    noise,status _ port         ;turn off noise LED
                bra     continue                    ;skip next line of code
yes _ noise _ err bclr    noise, status _ port        ;turn on noise LED
continue        jsr     put _ char                  ;transmit the received byte
*               bra     main _ loop                 ;and prepare for next reception.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* init——initialize port pins for sci operation and for driving LEDs;called by main  *
* input cond. ——none                                                                *
* output cond. ——TXD=output initialize to 1, RXD=input, noise LED=off, frame LED=off. *
* stack used——0 bytes                                                               *
* variables used——none                                                              *
* ROM used——15 bytes                                                                *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
init            bset    txd,serial _ port           ;init txd=1
                bset    txd,serial _ port+4         ;txd=output
                bclr    rxd,serial _ port+4         ;rxd=input
                bset    noise,status _ port         ;noise LED=off
                bset    noise,status _ prot+4       ;noise=output
                bset    frame,status _ port         ;frame LED=off
                bset    frame,status _ port+4       ;frame=output
                rts                                 ;exit(init)
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* get _ char——receive one byte of data from RXD pin;called by main              *
* input cond. ——RXD pin defined as an input pin                                 *
* output cond. ——char contains received data; X,ACC undefined;                  *
*               half carry=1(frame occured) or 0(no frame error);               *
*               Carry=1(noise and/or frame error occured)or 0(no noise).        *
* stack used——2 bytes                                                           *
* variables used——char: storage for received data (1 byte)                      *
*               count: temporary storage (1) byte                               *
* ROM used——63 bytes                                                            *
* * * * * * * * * *  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
get _ char      lda     #8                          ;[2] receiving 8 data bits
                sta     count                       ;[4] store value into RAM
                clrx                                ;[3] used to store noise data
get _ start _ bit brclr   rxd,serial _ port, *        ;[5] wait until rxd=1
                brset   rxd,serial _ port, *        ;[5] wait for start bit
```

```
                lda     $BAUD_SEL-3             ;[2] prepare for 1/2 bit delay
                bsr     delay_13a               ;[13a+12] execute delay routine
                bsr     get_bit                 ;[39] sample start bit
                lsra                            ;[3] noise bit->carry;
                                                ;    acc=filtered start bit
                bne     get_start_bit           ;[3] if false start, start over
                tsta                            ;[3] for timing purposes only
                tsta                            ;[3] for timing purposes only
                lda     #2*(BAUD_SEL-2)         ;[2] prepare for 1 bit delay
                bsr     delay_13a               ;[13a+12] execute delay routine
get_data_bits   bsr     get_bit                 ;[39] sample data bit
                rora                            ;[3] noise bit->carry
                rorx                            ;[3] carry->noise data reg
                rora                            ;[3] filtered data bit->carry
                ror     char                    ;[5] carry->char
                lda     #2*(BAUD_SEL-3)         ;[2] prepare for 1 bit delay
                bsr     delay_13a               ;[13a+12] execute delay routine
                tsta                            ;[3] for timing purposes only
                dec     count                   ;[5] bit received, dec count
                bne     get_data_bits           ;[3] loop if more bits to get
get_stop_bit    bsr     get_bit                 ;[39] sample stop bit
                lsra                            ;[3] noise bit->carry
                                                ;    acc=filtered stop bit
                sta     count                   ;[4] store stop bit in count
                bcc     yes_noise               ;[3] if noise, then branch
                txa                             ;[2] noise data→acc
                eor     char                    ;[3] XOR noise with char,
                beq     no_noise                ;[3] and if result=0
                                                ;    then no noise in data
                                                ;    reception
yes_noise       lda     #$08                    ;[2] set noise bit (half carry)
                add     #$08                    ;[2] by adding $8 to $8
no_noise        lda     count                   ;[3] retrieve stop data bit,
                coma                            ;[3] complement it,
                lsra                            ;[3] and shift it into carry
                                                ;    for frame error bit
                rts                             ;[6] exit (get_char)
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* get_bit——receive one bit of filtered data and noise info; called by get_char          *
* input cond. ——RXD pin defined as an input pin                                         *
* output cond. ——ACC= 000000dn, where d=filtered data, n=noise info                     *
* stack used——0 bytes                                                                   *
* variables used——none                                                                  *
```

```
* ROM used——17 bytes                                                         *
*****************************************************************************
get _ bit           clra                                ;[3] used to add sampled bits
                    brset    rxd,serial _ port,samp _ 1 ;[5] sample 1st bit into carry
samp _ 1            adc      #0                         ;[3] add it to acc
                    brset    rxd,serial _ port,samp _ 2 ;[5] sample 2nd bit into carry
samp _ 2            adc      #0                         ;[3] add it to acc
                    brset    rxd,serial _ port,samp _ 3 ;[5] sample 3rd bit into carry
samp _ 3            adc      #0                         ;[3] add it to acc
                    rts                                 ;[6] exit (get _ bit)
*****************************************************************************
* put _ char——transmit data byte in char out onto TXD pin; called by main    *
* input cond. ——TXD pin defined as an output pin and TXD=1;                  *
*          char contains byte to be tranmitted.                              *
* output cond. ——X,ACC,char=undefined;                                       *
* stack used——2 bytes                                                        *
* variables used——char: storage for transmitted data (1 byte)                *
* ROM used——31 bytes (35 if sending two stop bits)                           *
*****************************************************************************
put _ char          ldx      #9                         ;[2] be sending 8 data bits
                    clc                                 ;[2] clear carry for start bit
put _ data _ bits   bcc      send _ 0                   ;[3] if carry<>0,then
                    bset     txd,serial _ port          ;[5]   send out a 1
                    bra      jmp _ bit                  ;[3]   finished sending a 1
send _ 0            bclr     txd,serial _ port          ;[5] else send a 0
                    bra      jmp _ bit                  ;[3]   finished sending a 0
jmp _ bit           lda      #2*(BAUD _ SEL-1)-1        ;[2] prepare for a 1 bit delay
                    bsr      delay _ 13a                ;[13a+12] execute delay routine
                    tsta                                ;[3] for timing purposes only
                    ror      char                       ;[5] get next data bit to send
                    decx                                ;[3] one bit sent, so dec count
                    bne      put _ data _ bits          ;[3] loop if more bits to send
put _ stop _ bit    nop                                 ;[2] for timing purposes only
                    bset     txd,serial _ port          ;[5] send out a one
                    lda      #2*(BAUD _ SEL-1)          ;[2] prepare for a 1 bit delay
                    bsr      delay _ 13a                ;[13a+12] execute delay routine
* add the next two lines to guarantee sending two stop bits:
*                   lda      #2*(BAUD _ SEL-1)+1        ;[2] prepare for a 1 bit delay
*                   bsr      delay _ 13a                ;[13a+12] execute delay routine
                    rts                                 ;[6] exit (put _ char)

*****************************************************************************
* delay _ 13a——delay for 13*ACC+12 cycles; called get _ char and put _ char   *
```

```
* input cond. ——ACC set to appropriate value (13 * ACC+12 cycles)          *
* output cond. ——ACC=0                                                      *
* stack used——0 bytes                                                       *
* variables used——none                                                      *
* ROM used——7 bytes                                                         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
delay_13a       nop                     ;[2] this is a 13-cycle loop
                nop                     ;[2]
                tsta                    ;[3]
                deca                    ;[3] decrement loop count
                bne     delay_13a       ;[3] loop if count not zero
                rts                     ;[6] exit (delay_13a)
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*               Interrupt and Reset vectors for Main Routine                *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                org     RESET
                fdb     main
```

5.3 J系列单片机与EEPROM端口

在这一节中，我们介绍J系列单片机与EEPROM芯片9356/9366端口的方法。在实际应用中，经常需要保存一些用户设定修改的参数，以便掉电以后仍能保存。而J系列内部无EEPROM内存，这就需要外接EEPROM芯片。93××系列的EEPROM芯片是经常使用的芯片，采用串行端口方式进行读/写。9356内部具有128个16位EEPROM单元，9366具有256个16位EEPROM单元。这些芯片体积小，8只脚封装，容量大，适合各种应用场合。

由于这类芯片是通过串行端口与外部交换数据的，而J系列不支持这样的端口，因此，只有通过软件控制和仿真这种端口以读取或存储外部EEPROM芯片数据。

一、9356硬件端口

其串行端口主要由4根信号线组成：

CS——片选；

SK——同步时钟；

DO——串行数据输出；

DI——串行数据输入。

对该芯片的读/写操作都是通过输入/输出一系列串行数据位实现的。9356芯片的一组串行数据结构为：

操作码+地址+数据

其中，操作码3位，地址8位，数据16位。

3位操作码可以构成7种命令：

READ——从指定的地址单元读数据；

WRITE——将 16 位数据写到指定位置处；

WRAL——将某 16 位数据写到其内部所有空间；

ERASE——删除指定地址单元的数据；

ERAL——删除所有数据；

EWEN——允许删除与写数据；

EWDS——禁止删除与写数据。

各操作命令格式如表 5-1 所示。

表 5-1　9356 芯片的操作命令格式

命　令	操　作　码	地　址	数　据
READ	110	A7～A0	
EWEN	100	11××××××	
ERASE	111	A7～A0	
ERAL	100	10××××××	
WRITE	101	A7～A0	D15～D0
WRALL	100	01××××××	D15～D0
EWDS	100	00××××××	

了解了 9356 的端口协议后，就可以在单片机的控制程序中用软件实现数据传送的协议了。用 4 个 I/O 引脚与其 4 个控制脚相连，单片机通过软件使 4 个 I/O 引脚仿真实现其对应的功能。如图 5-14 所示电路，用 PA0～PA3 来仿真实现 CS，SK，DI，DO 的功能。

二、软件框图

下面给出了对 9356 芯片操作的程序，它主要由以下几个部分组成：

主程序流程如图 5-15 所示，用于初始化 I/O 端口，测试与 9356 读/写操作。它首先在 9356 的 00H 地址单元写入数据 AA55H，在 20H 单元写入数据 1234H；然后再执行读操作，先读 00H 单元的数据并存放在 TEST1 与 TEST2 中，再读 20H 单元的数据并存放在 TEST3，TEST4 中；最后测试结果，将结果通过 LED 发光二极管显示。

J9356 _ ERASE 子程序，实现删除一个单元功能，如图 5-16 示。

J9356 _ ERAL 子程序，用于删除全部单元数据，如图 5-16 示。

J9356 _ READ 子程序，用于读一个字数据，如图 5-17 示。

J9356 _ EWDS 子程序，实现禁止写功能，如图 5-18 示。

J9356 _ EWEN 子程序，实现允许写功能，如图 5-18 示。

J9356 _ WRITE 子程序，实现在某单元写一字节数据功能，如图 5-19 示。

J9356 _ WRAL 子程序，实现在所有单元写一个字数据的功能，如图 5-19 示。

J9356 _ WR _ OP 子程序，实现发送操作码功能，如图 5-20 示。

J9356 _ WR _ ADDR 子程序，实现发送地址功能，如图 5-20 示。

J9356 _ WR _ DATA 子程序，实现发送数据功能，如图 5-20 示。

J9356 _ RD _ DATA 子程序，实现读出 16 位数据的功能，如图 5-20 所示。

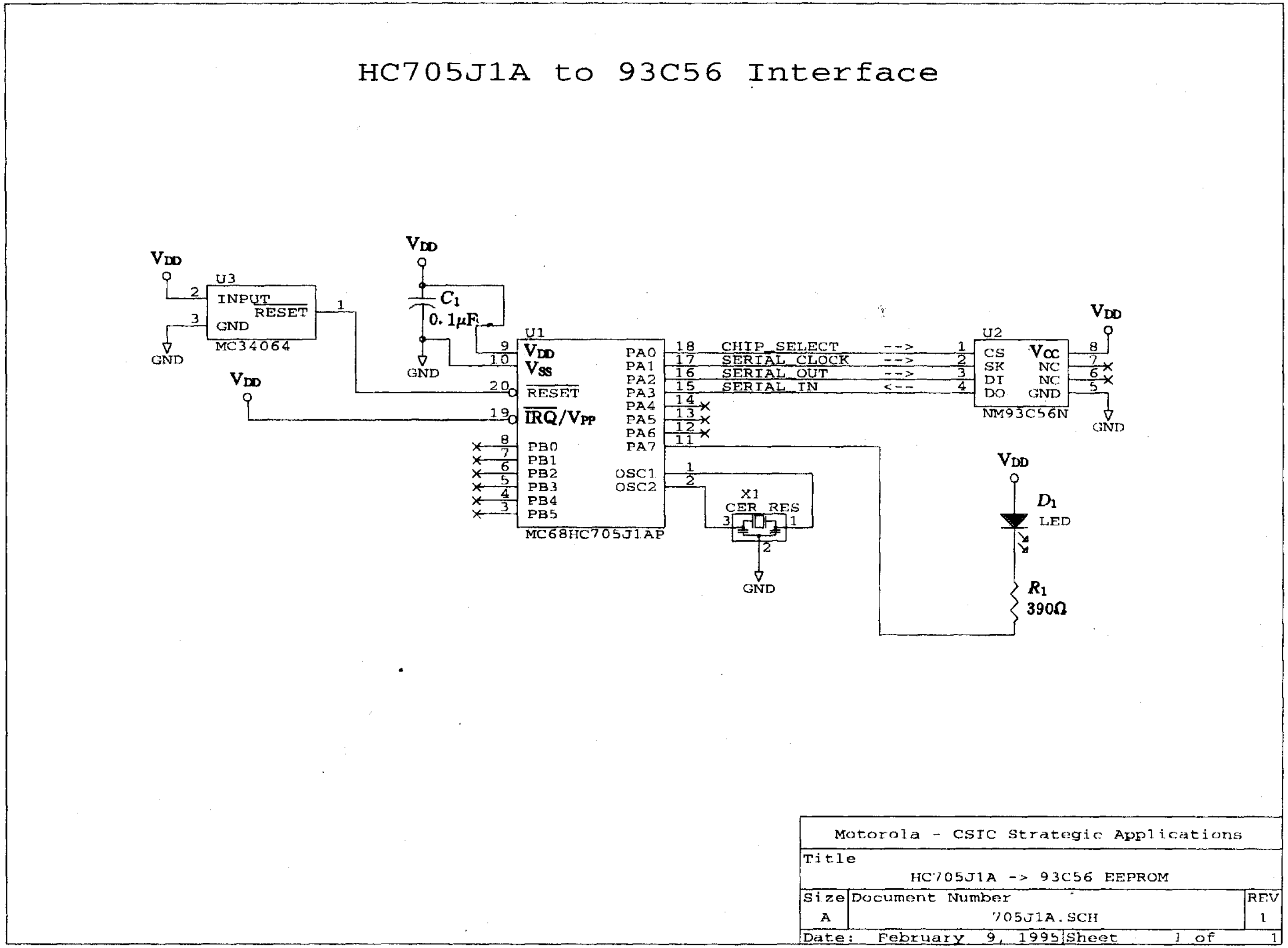


图 5-14 HC705J1A 与 93C56 的端口电路

J9356_WAIT 子程序，实现等待功能，如图 5-20 所示。

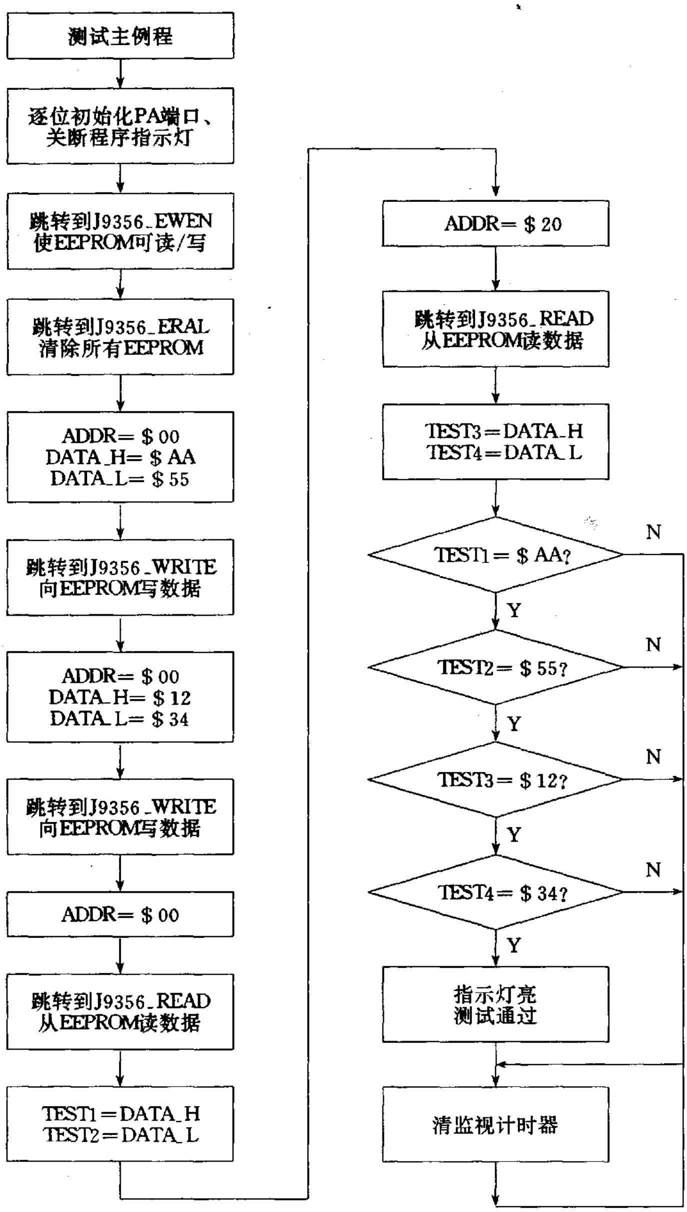


图5-15　主程序结构流程图

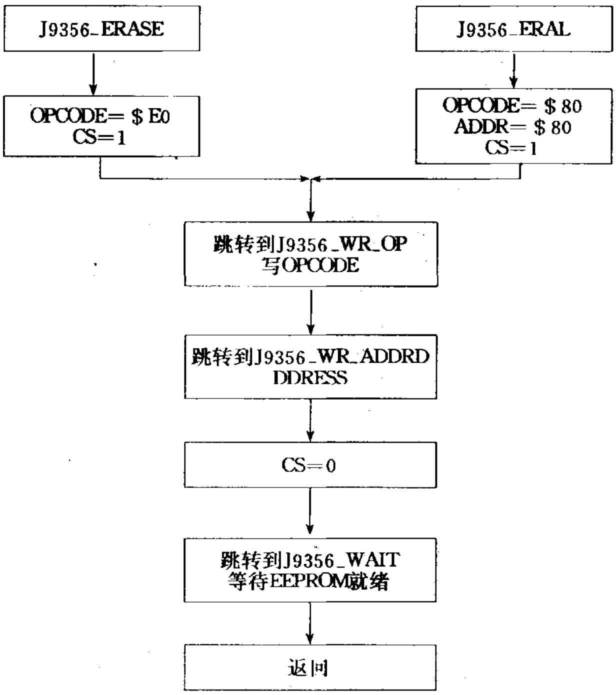


图5-16 删除数据子程序流程

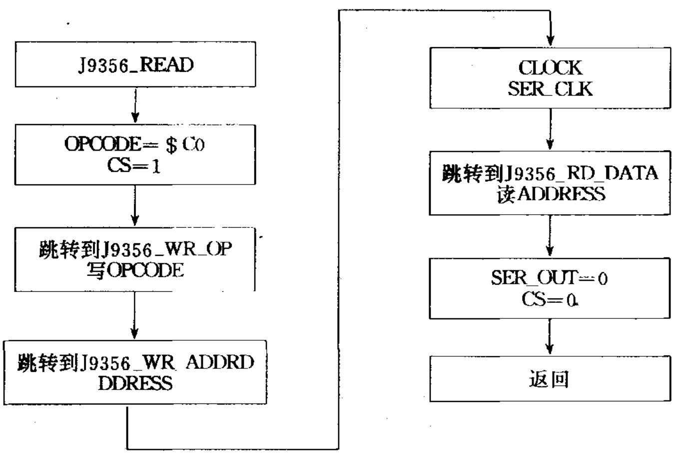


图5-17 读一个字数据的子程序流程图

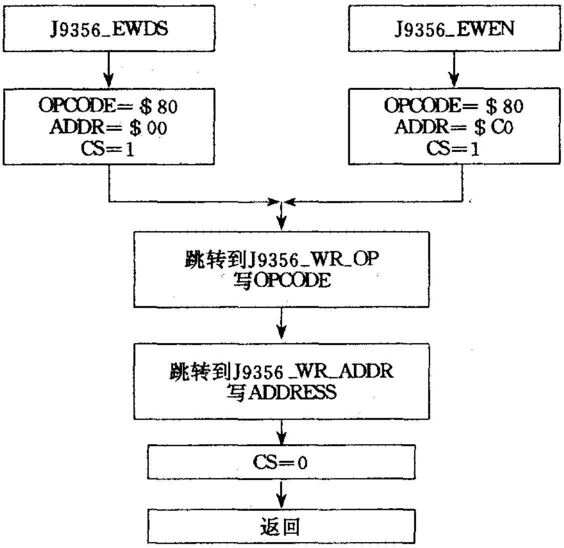


图5-18　实现禁止／允许写操作的子程序流程

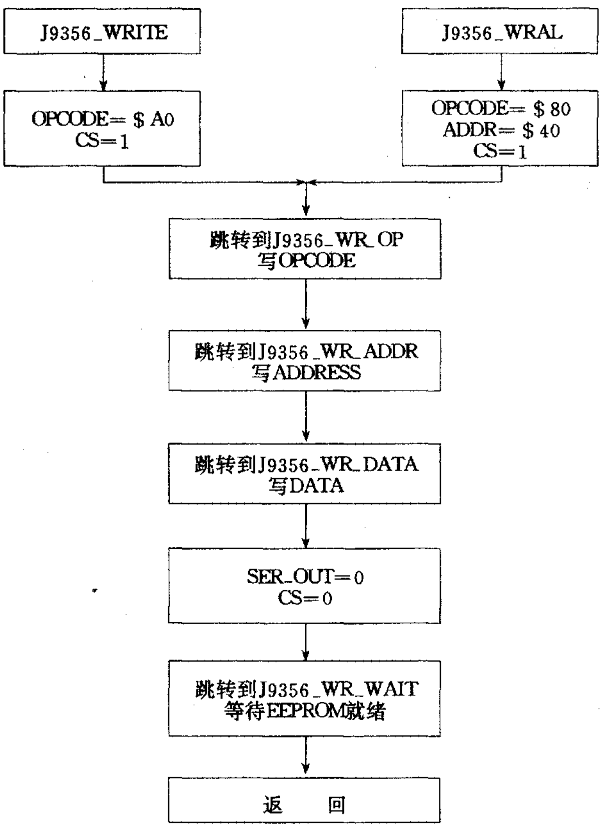


图5-19　写入数据的子程序流程

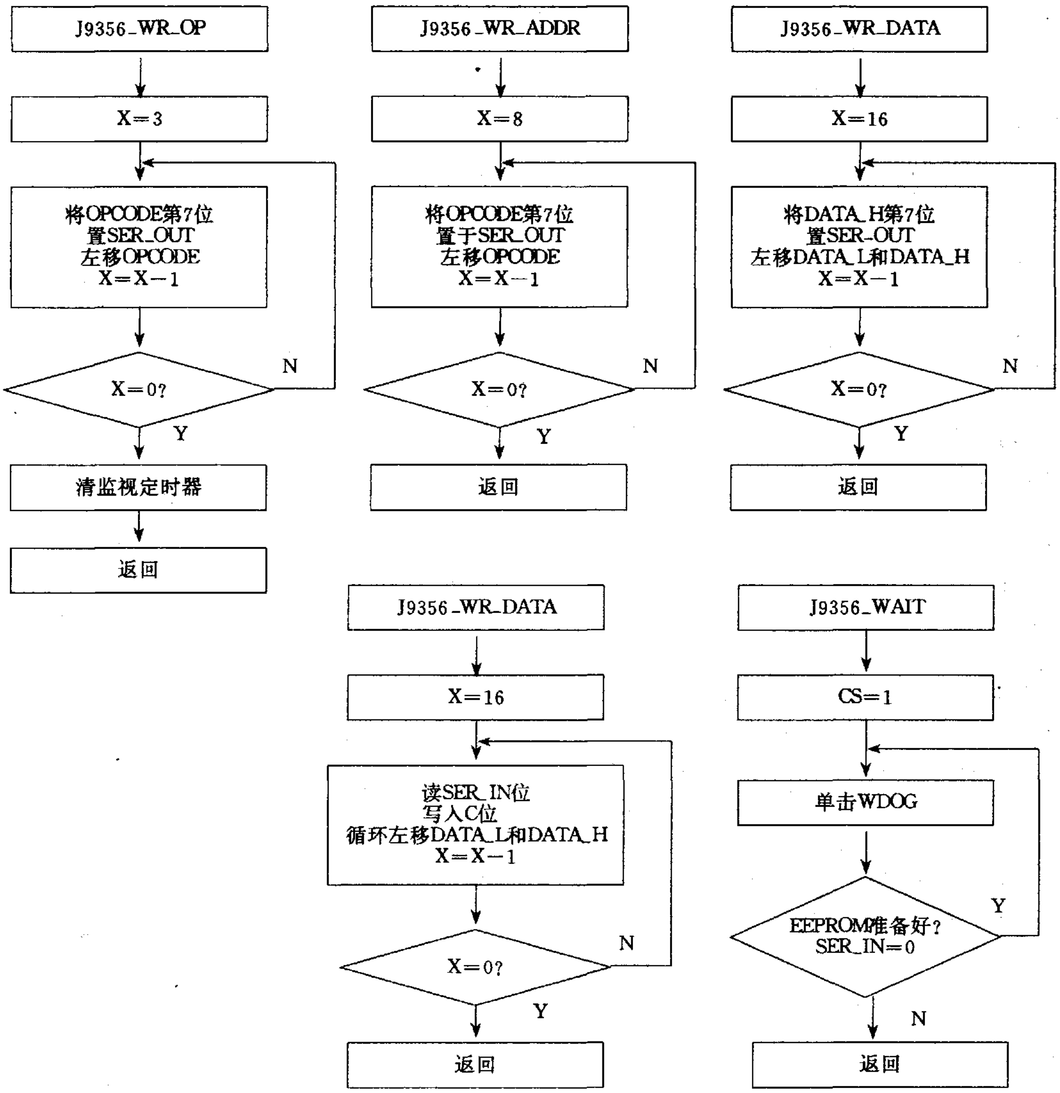


图5-20　发送数据、读出数据及等待子程序的流程

三、程序清单

```
#nolist
#INCLUDE 'HC705J1A.FRK'                    ;Include the equates for the HC705J1A
                                           ;so that all labels can be used.
#list
****************************************
*        MOR Bytes Definitions for Main Routine        *
****************************************
          org      MOR                      ;COP enabled, osc resistor enabled
          db       $21                      ;If used on a mask ROM part,
```

```
                                                ; be sure to specify this option.
****************************************************
*           Equates and RAM Storage                 *
****************************************************
CS              equ     0                       ;bit # for chip select
SER_CLK         equ     1                       ;bit # for serial clock
SER_OUT         equ     2                       ;bit # for serial data out
SER_IN          equ     3                       ;bit # for serial data in
***     RAM storage variables     ***

                org     RAM                     ;start of static RAM at $C0
OPCODE          rmb     1                       ;command byte
ADDR            rmb     1                       ;EEPROM address byte
DATA_H          rmb     1                       ;MSByte of data
DATA_L          rmb     1                       ;LSByte of data
TEST1           rmb     1                       ;test byte #1
TEST2           rmb     1                       ;test byte #2
TEST3           rmb     1                       ;test byte #3
TEST4           rmb     1                       ;test byte #4

****************************************************
*                Program Initialization             *
* This section sets up the port for bit banging
* To prevent floating inputs and associated high current draw.    *
* the HC705J1A has pulldown devices on all I/O pins. This         *
* initialization should enable these pulldowns on unused I/O      *
* pins. RESET_enables the pulldowns, so no code is required.      *
****************************************************

                org     EPROM
J9356_START     lda     #$80                    ;init portA
                sta     PORTA
                sta     COPR                    ;kick the wdog
                lda     #$87                    ;init I/O of port A
                sta     DDRA

****************************************************
*             J1A_9356 Main Program Loop            *
* It then runs through the test routine to check for             *
* proper serial transmission. The LED is lit if the test passes. *
****************************************************

  ***     Enable erase/write mode of EEPROM
                jsr     J9356_EWEN              ;call ewen routine
```

```
* * *       Erase all EEPROM memory map
                jsr       J9356_ERAL              ;call eral routine

* * *       Write $AA55 to $00
                lda       #$00                    ;load address
                sta       ADDR
                lda       #$AA                    ;load data byte high
                sta       DATA_H
                lda       #$55                    ;load data byte low
                sta       DATA_L
                jsr       J9356_WRITE             ;call write routine

* * *       Write $1234 to $20
                lda       #$20                    ;load address
                sta       ADDR
                lda       #$12                    ;load data byte high
                sta       DATA_H
                lda       #$34                    ;load data byte low
                sta       DATA_L
                jsr       J9356_WRITE             ;call write routine

* * *       Read $00
                lda       #$00                    ;load address
                sta       ADDR
                jsr       J9356_READ              ;call read routine
                lda       DATA_H
                sta       TEST1                   ;store away data_h to test1
                lda       DATA_L
                sta       TEST2                   ;store away data_l to test2

* * *       Read $20
                lda       #$20                    ;load address
                sta       ADDR
                jsr       J9356_READ              ;call read routine
                lda       DATA_H
                sta       TEST3                   ;store away data_h to test3
                lda       DATA_L
                sta       TEST4                   ;store away data_l to test4

* * *     Check results of write and read, light LED if good
J9356_CKSUM     lda       TEST1                   ;check test1
                cmpa      #$AA
                bne       J9356_BRANCH            ;branch if no good, no LED
                lda       TEST2                   ;check test2
                cmpa      #$55
```

```
                bne     J9356_BRANCH        ;branch if no good, no LED
                lda     TEST3               ;check test3
                cmpa    #$12
                bne     J9356_BRANCH        ;branch if no good ,no LED
                lda     TEST4               ;check test4
                cmpa    #$34
                bne     J9356_BRANCH        ;branch if no good, no LED
                bclr    7,PORTA             ;EEPROM write and read is good
                                            ;light LED
J9356_BRANCH    clra
                sta     COPR                ;kick the wdog
                bra     J9356_BRANCH

***************************************
*            EEPROM Command SubRoutines                                          *
* These 7 subroutines execute each of the 7 commands that the EEPROM will respond to *
***************************************

***     EWEN——subroutine to enable write/erase ***************
J9356_EWEN      lda     #$80                ;load opcode
                sta     OPCODE
                lda     #$C0                ;load address
                sta     ADDR
                bset    CS,PORTA            ;CS line is high
                jsr     J9356_WR_OP         ;write opcode
                jsr     J0356_WR_ADDR       ;write address
                jsr     J9356_WR_DATA       ;write data
                bclr    SER_OUT, PORTA
                bclr    CS,PORTA            ;CS line is low
                rts                         ;return

***     EWDS——subroutine to disable write/erase ************
J9365_EWDS      lda     #$80                ;load opcode
                sta     OPCODE
                clr     ADDR                ;load addr
                bset    CS,PORTA            ;CS line is high
                jsr     J9356_WR_OP         ;write opcode
                jsr     J9356_WR_ADDR       ;write address
                bclr    CS,PORTA            ;CS line is low
                rts                         ;return

***     WRITE——subroutine to write EEPROM *************
J9356_WRITE     lda     #$A0                ;load opcode
                sta     OPCODE
                bset    CS,PORTA            ;CS line is high
```

```
          jsr     J9356_WR_OP          ;write opcode
          jsr     J9356_WR_ADDR        ;write addres
          jsr     J9356_WR_DATA        ;write data
          bclr    SER_OUT,PORTA
          bclr    CS,PORTA             ;CS line is low
          jsr     J9356_WAIT           ;wait until EEPROM is ready
          rts                          ;return

 * * *      WRAL——subroutine to write all EEPROM * * * * * * * * * * *
J9356_WRAL    lda     #$80             ;load opcode
              sta     OPCODE
              lda     #$40             ;load addr
              sta     ADDR
              bset    CS,PORTA         ;CS line is high
              jsr     J9356_WR_OP      ;write opcode
              jsr     J9356_WR_ADDR    ;write address
              jsr     J9356_WR_DATA    ;write data
              bclr    SER_OUT RORTA
              bclr    CS,PORTA         ;CS line is low
              jsr     J9356_WAIT       ;wait until EEPROM is ready
              rts                      ;return

 * * *    READ——subroutine to read EEPROM * * * * * * * * * * * * * * *
J9356_READ    lda     #$C0             ;load opcode
              sta     OPCODE
              bset    CS,PORTA         ;CS line is high
              jsr     J9356_WR_OP      ;write opcode
              jsr     J9356_WR_ADDR    ;write address
              bset    SER_CLK,PORTA    ;clock the EE
              bclr    SER_CLK,PORTA
              jsr     J9356_RD_DATA    ;read data
              bclr    SER_OUT,PORTA
              bclr    CS,PORTA         ;CS line is low
              rts                      ;return

 * * *    ERASE——subroutine to erase EEPROM * * * * * * * * * * * * * * *
J9356_ERASE   lda     #$E0             ;load opcode
              sta     OPCODE
              bset    CS,PORTA         ;CS line is high
              jsr     J9356_WR_OP      ;write opcode
              jsr     J9356_WR_ADDR    ;write address
              bclr    CS,PORTA         ;CS line is low
              jsr     J9356_WAIT       ;wait until EEPROM is ready
              rts
```

```
 * * *      ERAL——subroutine to erase all EEPROM * * * * * * * * * * * * * *
J9356_ERAL          lda     #$80                    ;load opcode and addr
                    sta     OPCODE
                    sta     ADDR
                    bset    CS,PORTA                ;CS line is high
                    jsr     J9356_WR_OP             ;write opcode
                    jsr     J9356_WR_ADDR           ;write address
                    bclr    CS,PORTA                ;CS line is low
                    jsr     J9356_WAIT              ;wait until EEPROM is ready
                    rts

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                       EEPROM Supporting SubRoutines                          *
* These subroutines support the functions called from the Command subs         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 * * *      Subroutine to write 3 bit opcode * * * * * * * * * * * * * * * * * *
J9356_WR_OP         ldx     #3T                      ;init counter for LOOP1

 *          Write to the serial output pin
J9356_LOOP1         brclr   7,OPCODE,J9356_L1_2      ;if opcode bit7=0,goto L1_2
                    bset    SER_OUT,PORTA            ;ser_out=1
                    bra     J9356_L1_3               ;goto L1_3
J9356_L1_2          bclr    SER_OUT,PORTA            ;ser_out=0

 *          Clock the serial clock pin
J9356_L1_3          bset    SER_CLK,PORTA            ;ser_clk=1
                    bclr    SER_CLK,PORTA            ;ser_clk=0
                    asl     OPCODE                   ;rotate the opcode
                    decx                             ;decrease counter loop
                    bne     J9356_LOOP1              ;is LOOP1 finished?
                    clra
                    sta     COPR                     ;kick the wdog
                    rts                              ;return

 * * *        Subroutine to write 8 bit address * * * * * * * * * * * * * * * * *
J9356_WR_ADDR       ldx     #8T                      ;init counter for LOOP2

 *          Write to the serial output pin
J9356_LOOP2         brclr   7,ADDR,J9356_L2_2        ;if addr bit7=0,goto L2_2
                    bset    SER_OUT,PORTA            ;ser_out=1
                    bra     J9356_L2_3               ;goto L2_3
J9356_L2_2          bclr    SER_OUT,PORTA            ;ser_out=0
```

```
*         Clock the serial clock pin
J9356_L2_3          bset    SER_CLK,PORTA             ;ser_clk=1
                    bclr    SER_CLK,PORTA             ;ser_clk=0
                    asl     ADDR                      ;rotate the addr
                    decx                              ;decrease counter loop
                    bne     J9356_LOOP2               ;is LOOP2 finished?
                    rts                               ;return

***Subroutine to write 16 bit data********************
J9356_WR_DATA       ldx     #16T                      ;init counter for LOOP4

*         Write the serial output pin with data
J9356_LOOP4         brclr   7,DATA_H,J9356_L4_2       ;if addr bit7=0,goto L4_2
                    bset    SER_OUT,PORTA             ;ser_out=1
                    bra     J9356_L4_3                ;goto L4_3
J9356_L4_2          bclr    SER_OUT,PORTA             ;ser_out=0

*         Clock the serial clock pin
J9356_L4_3          bset    SER_CLK,PORTA             ;ser_clk=1
                    bclr    SER_CLK,PORTA             ;ser_clk=0
                    asl     DATA_L                    ;rotate the DATA_L
                    rol     DATA_H                    ;rotate the DATA_H
                    decx                              ;decrease counter loop
                    bne     J9356_LOOP4               ;is LOOP4 finished?
                    rts                               ;return

***       Subroutine to read 16 bit data*******************
J9356_RD_DATA       ldx     #16T                      ;init counter for LOOP3

*         Read the serial input pin
J9356_LOOP3         brclr   SER_IN,PORTA,J9356_L3     ;carry bit=serial in
J9356_L3            rol     DATA_L                    ;rotate left result
                    rol     DATA_H

*         Clock the serial clock pin
                    bset    SER_CLK,PORTA             ;ser_clk=1
                    bclr    SER_CLK,PORTA             ;ser_clk=0
                    decx                              ;decrease counter loop
                    bne     J9356_LOOP3               ;is LOOP3 finished?
                    rts                               ;return

*         Wait until write cycle is over
J9356_WAIT          bset    CS,PORTA                  ;CS line is high
J9356_W2            clra
                    sta     COPR                      ;kick the wdog
```

```
        brclr   SER_IN,PORTA,J9356_W2
        bclr    CS,PORTA                ;CS line is low
        rts                             ;return

****************************************
*     Interrupt and Reset vectors for Main Routine     *
****************************************

        org     RESET
        fdb     J9356_START
```

第六章　MC68HC05K 系列单片机的应用

MC68HC05K0/K1 单片机是 MOTOROLA 公司生产的单片机家族中档次最低的廉价单片机。由于它们的引脚只有 16 只，所以，只能用于控制功能较少的系统中。由于 K 系列单片机的内存容量很小，用户程序存储器小于 1 KB，故而程序的能力也是有限的。但是，这种单片机在简单的控制仪器、家用电器、玩具、计算机外部设备等方面有着很广的用途。因为在一些简单控制的场合，并不需要很强功能的控制器，而 K 系列单片机刚好能承担这种任务。

这一章将介绍 MC68HC05K0/K1 单片机在实际中的应用，包括它在家用电器、遥控器、温度检测器、微工件抛光机等方面的应用。从中可以体会到 MC68HC05K0/K1 单片机在实际应用中是十分有用的。有时为了实现一些功能，在使用结构简单的 K 系列单片机时需要更多硬件和软件设计技巧。

6.1　68HC05K0 控制的蒸炖煲

蒸炖煲是一种在亚洲流行的家用电器，它是以水为媒介，对炖盅中的食品进行加热。由于水温最高为 100 ℃，蒸汽的温度则高于 100 ℃，在蒸炖煲中是以水温和蒸汽的温度对煲内的炖盅进行加热，使盅内的食品经过一定时间的加热而释放出其中的营养物质形成炖汁。由于煲内温度一般为 100 ℃～120 ℃左右，所以能保持食品的良好营养成分，又能使其营养物质溶于水中，有利于吸收。这种蒸炖煲基于中医的医学原理，为亚洲人进补提供了一种方便的煮炖工具。

蒸炖煲可以炖鸡、人参、各种补品和野味，也可以煮米饭或粥，还可以加热冷了的食品，所以很受人们的欢迎。

蒸炖煲要求能控制蒸炖过程中的蒸汽产生量、蒸炖时间、预设时间、干水报警、蒸炖结束报警等，要完成这些功能需要单片机进行控制。

这一节中将介绍 68HC05K0 控制蒸炖煲的方法。

6.1.1　蒸炖煲的结构及蒸炖原理

蒸炖煲是一种既有蒸煮功能又有炖煮功能的电热烹煮工具。在使用中通常是两种功能同时加以使用的。

蒸炖煲由底座、加热盘、煲体、煲盖以及炖盅组成，结构如图 6-1 所示。

底座是蒸炖煲的底部支撑部件，它一方面用于支撑整个蒸炖煲，另外还把加热盘和底分隔开来，以防止加热盘的热量直接传到底座的底上而浪费热量或损伤与蒸炖煲接触的台面等。

加热盘是以电为能源的电热丝加热器。加热盘内部有电热丝，电热丝通过一些绝缘的氧化物粉剂和盘体相连。盘体是金属铝的铸盘，它把加热丝包在内部，由于盘体与加热丝之间有绝缘氧化物粉剂填充，故而盘体不会带电，但可以传导加热丝产生的热量。加热盘产生的热量直接用于对煲体的水加热并产生蒸汽。

煲体用于盛放水及蒸炖盅。煲体的水直接由加热盘加热，然后水再把热量传给蒸炖盅。煲

体中的水在到达沸点之后会产生蒸汽，加热盘的功率越大，产生的蒸汽越多。控制加热盘的功率可以控制蒸汽的产生量。蒸汽的温度高于 100 ℃，它同时也会对蒸炖盅加热。

煲盖用于开启时放入或取出炖盅，当煲盖盖上时，能有效地防止蒸汽泄漏。煲盖越密封，则蒸汽泄漏越少，那么蒸炖煲的加热效率越高。但煲盖不能盖得太密，否则当加热盘功率足够大时，蒸汽难以泄漏会使煲内压力过高而产生危险。煲盖一般以自然方式盖上，当蒸汽压力较高时，煲盖会被托起自动泄漏一部分蒸汽，从而保证了蒸炖的高度安全性。

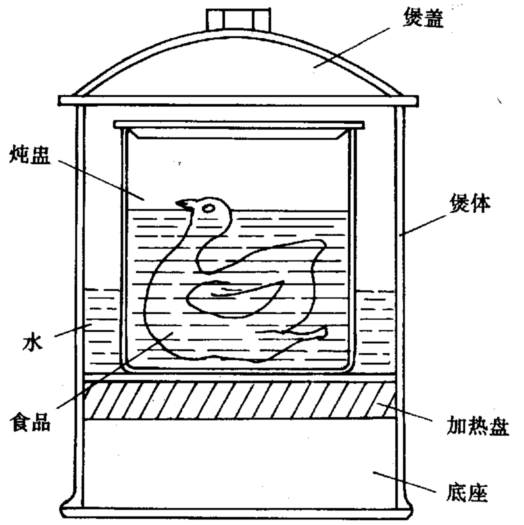


图 6-1　蒸炖煲结构

炖盅用来盛放食品，被炖的食品放在炖盅内部，依靠水和蒸汽的热量对盅体的作用使炖盅内的食品被缓缓加热，从而产生一种特殊的烹煮效果，炖出美味和松软的食品。

在蒸炖煲中，热量的来源是加热盘，而消耗热量的有食品、环境和蒸汽泄漏。热量平衡方程可用下式表示：

$$Q_e = Q_1 + Q_2 + Q_3 \tag{6-1}$$

其中，Q_e——加热盘产生的热量；

Q_1——食品吸收的热量；

Q_2——环境消耗的热量，包括煲体、炖盅所吸收的热量和外界环境吸收的热量；

Q_3——蒸汽泄漏的热量。

在式(6-1)中，当 Q_e 增加时，并非是 Q_1,Q_2 和 Q_3 按比例同时增加。由于 Q_e 增加会导致蒸汽增加，故而蒸汽压力上升，通过煲盖泄漏出去的也增加，这种泄漏的增加比 Q_1,Q_2 的增加要大得多，因此所造成的热量浪费上升。为了使热量减少浪费，就要适当控制蒸汽的量。一般要求 Q_e 在煲内的水沸腾之后能保持在一定的水平，它使水刚能沸腾而又不致于产生过多蒸汽，从而维持蒸炖过程的正常进行。

6.1.2　蒸炖煲的工作要求及控制框图

为了控制蒸炖煲在正常工作时节省能源，同时，用户能监视其工作状态，因此要求蒸炖煲能有如下一些工作性能。

1. 预置时间设定

这是一种方便用户操作的功能。当用户准备蒸炖食品时，可以考虑食品的类型而确定其蒸炖时间的长短。如果希望在某个时间开始蒸炖，过一定时间进食，就可以提早预置时间，令蒸炖煲延时一定时间后开始工作。例如，用户晚上 10 时把食品准备好，希望蒸炖煲早上 5 时开始蒸炖，以准备蒸炖 2 个小时后于 7 时食用，则可以预置时间为 7 h。

2. 工作时间设定

这是设定蒸炖煲的实际蒸炖时间。例如，要蒸炖 2 h，则可以设定工作时间为 2 h。

3. 工作显示

当蒸炖煲进入蒸炖状态时，显示此时的工作状态，表示蒸炖煲正处于加热蒸炖过程中。

4. 半功率控制

在蒸炖煲开始加热时，煲内的水是凉的，所以，要用全功率进行加热，以加快水的温度上升，快些达到沸腾状态。在水到达沸腾状态后，只要有一定的加热功率就可以维持水的沸腾状态。这时，只需半功率加热即可，故要控制加热盘处于半功率工作状态。

5. 干水报警

这是一种安全要求。当煲内的水汽化并蒸发到一定程度，就会慢慢出现干水的情况。这时再加热就会产生高温，损坏蒸炖煲，或发生失火危险。所以，要对干水情况进行检测并予以报警。

6. 蒸炖结束报警

蒸炖过程结束时，蒸炖煲会发出特殊节奏的声光报警，提醒用户蒸炖过程已经结束。这是一种有利于用户及时进食的提示性特性。

蒸炖煲电脑控制器的结构框图如图 6-2 所示，包括电源、信号检测、显示、控制输出和按键输入等五个部分。

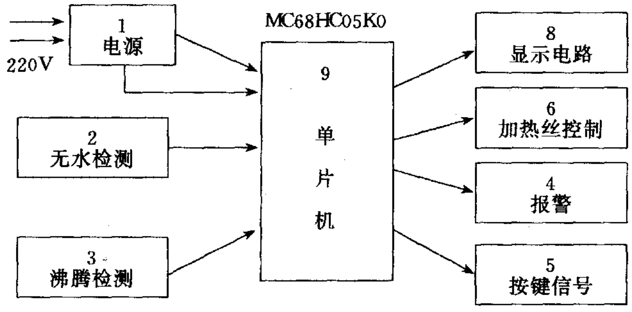


图6-2　蒸炖煲控制器框图

电源部分由变压器和有关稳压器件组成，其作用是向 MC68HC05K0 提供 5 V 直流稳压电源，同时向 MC68HC05K0 提供过零电压信号。

信号检测部分包括两种功能：无水检测和沸腾检测。无水检测用于判断炖煲是否处于干水状态，如果处于干水状态则不能再进行加热，否则会破坏煲体。沸腾检测用于判断煲体内的水是否已沸腾，如果已沸腾则煲体进入了产生蒸汽的阶段。这时要对蒸汽量进行控制，以求得好的蒸炖效果。

显示部分由时间显示和工作指示部分组成。时间显示由两个 7 段显示管显示，一个用于显示预置时间，一个用于显示工作时间，单位为 h(小时)。预置时间是指延时时间，延时时间最长为9 h，用户可以设定蒸炖煲在 1 h～9 h 之后才开始工作。工作时间最长为 5 h，用户可以设定蒸炖时间为 1 h～5 h。工作指示是一个发光二极管，只有加热丝工作时它才发光。

输出控制部分有两个，一个是加热丝控制，一个是蜂鸣器控制。加热丝控制是由继电器驱动的。当继电器吸合时加热丝工作，继电器断开时加热丝停止工作。加热丝工作时有全功率和半功率两种状态，在水未达沸腾状态时，采用全功率加热；在水沸腾时采用半功率加热。蜂鸣器是一个频率为 2 kHz 的陶瓷喇叭，单片机输出 2 kHz 方波时蜂鸣器会产生鸣叫。

按键信号是由三个按键产生的，因此有三种不同的按键信号。控制器的面板如图 6-3 所示。从图中可知有按键 N_1，N_2，N_3，其中，N_2 是预置时间按键，它可以对 7 段数码管 L_1 设置 1～9 这 9 个数字；N_3 是工作时间设定按键，可以对 7 段数码管 L_2 设置 1 h～5 h的工作时间；N_1 是工作启动按键，当按一下 N_1 时，则表示进入工作过程。

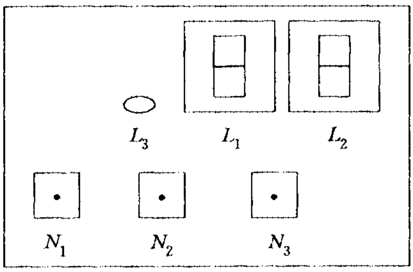


图6-3　蒸炖煲控制面板

6.1.3　电路工作原理

在蒸炖煲电脑控制器中，电源电路、检测电路、显示电路、输出控制电路和按键输入电路都是围绕经济性及可靠性而研制的，它们有着十分有效的作用。

1. 电源稳压及过零电路

电源稳压及过零电路如图 6-4 所示，它由变压器 *TF*、整流桥 *RB*、二极管 *D*、稳压集成电路 7805、整形电路和电容 C_1～C_4 组成。

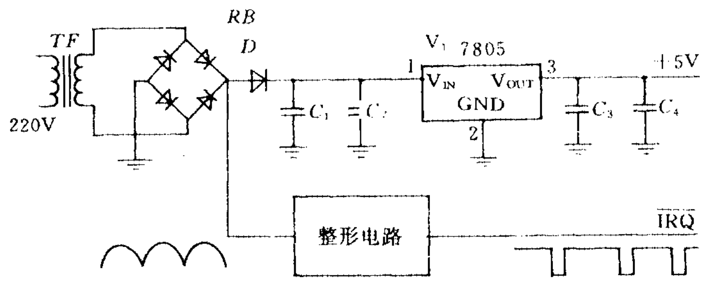


图 6-4　电源及过零检测电路

变压器 *TF* 把 220 V 的交流电压降压后成为 8 V 低压，通过整流桥 *RB* 整流后成了脉动直流电压，经过二极管 *D* 之后由 C_1，C_2 滤波得到无波纹的直流电压，再经稳压集成电路 7805 稳压成为＋5 V 电压输出。C_3，C_4 用于负载突变时的滤波。二极管 *D* 用于隔离脉动直流电压和滤波后的直流电压。脉动直流电压经整形电路整形后产生 10 ms 负脉冲过零信号，这个过零脉冲信号用于时间计数。

2. 无水和沸腾检测电路

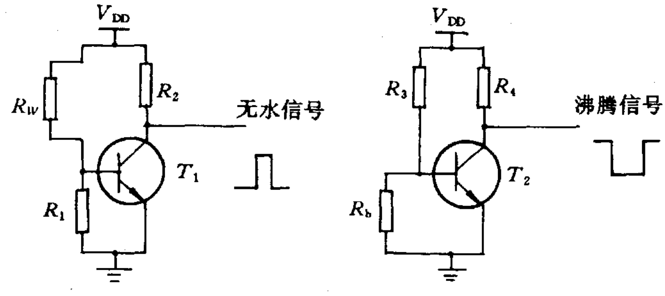
无水检测电路由水电阻和放大电路组成，结构如图 6-5(a)所示。当炖煲中有水时，水电阻 R_W 较小；而无水时，则水电阻为无穷大。水电阻的阻值和电阻 R_1 的阻值对电源电压进行分配，产生放大器的输入信号。有水时，晶体管 T_1 输出高电平。

沸腾检测电路由热敏电阻 R_b 和放大电路组成，结构如图 6-5(b)所示。当水温达到 100 ℃时，热敏电阻 R_b 的阻值突然增大，且阻值极大。所以，在未沸腾时，晶体管 T_2 输出高电平；而在沸腾时，T_2 输出低电平。一般对于 R_b 应选择其居里点为 98 ℃左右的较为恰当。

3. 显示电路

显示电路由两个 7 段发光二极管和一个光二极管组成，如图 6-6 所示。它们是一般通用的显示电路。

显示码从端口 PA0～PA6 直接通过限流电阻送到 7 段数码管的阴极，而数码管的阳极则



(a) (b)

图 6-5　无水和沸腾检测电路

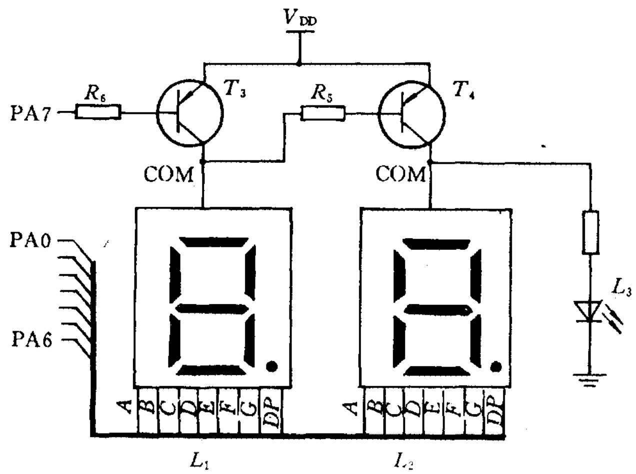


图 6-6　显示电路

由 PA7 所连的两个晶体管控制。当 PA7 输出低电平时，T_3 导通，T_4 截止，这时选中数码管 L_1；当 PA7 输出高电平时，T_3 截止，T_4 导通，这时选中数码管 L_2。只要把显示码从 PA0～PA7 送出，同时进行同步数码管选择，就可以以扫描方式进行显示。

4. 输出电路

输出控制电路由加热丝控制电路及报警控制电路组成，如图 6-7 所示。

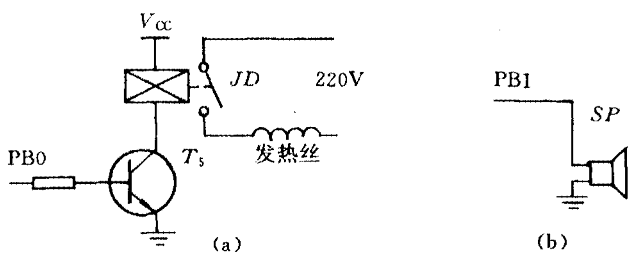


图 6-7　输出控制电路

输出加热丝控制电路是一个简单功率放大电路，由晶体管 T_5 以及继电器 JD 组成。当输

出端口 PB0 输出高电平时，T_5 导通，继电器 JD 吸合，使加热丝工作；反亦反之。如图 6-7(a)所示。

报警控制电路极其简单，单片机的输出端口 PB1 直接和陶瓷喇叭 SP 连接，只要从 PB1 输出的信号频率是陶瓷喇叭的固有频率，那么就可以产生最响的蜂鸣声。如图 6-7(b)所示。

5. 按键输入电路

按键输入电路有三个，分别为 N_1,N_2,N_3，如图 6-8 所示。当 N_1 按下时，PA7 输入低电平，表示启动炖煲工作。当 N_2 按下时，PB0 输入低电平，这时对数码管 L_1 置数。当 N_3 按下时，PB1 输入高电平，这时对数码管 L_2 置数。

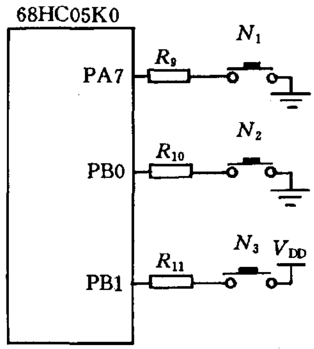


图 6-8 按键输入电路

6.1.4 控制软件及操作状态

控制软件是根据炖煲的工作需要和功能而编制的。控制软件要求能完成如下功能：

(1) 对炖煲的无水状态报警，以防止产生干煮现象而损坏炖煲；

(2) 能实现预置延时时间，设定蒸炖工作时间；

(3) 能执行所预置的时间延时，并按设定的工作时间工作；

(4) 能实现扫描显示；

(5) 能执行沸腾后的减小功率功能；

(6) 能以特殊方式显示工作状态。

控制软件流程如图 6-9 所示，由主程序和过零中断子程序组成。

过零中断子程序在电源交流电压过零时产生中断，其功能主要有两个：进行显示扫描和用于计时。由于电源是 50 Hz 的交流信号，故每 10 ms 存在一次过零时刻，故而过零子程序每 10 ms 执行一次。对于 7 段发光二极管显示器 L_1,L_2，每秒可以显示 50 次，每 10 ms 扫过一个，所以，可以实现无闪烁显示。在过零时刻中断时，中断过程可对内部存储单元进行计数计时。由于每 10 ms 产生一次中断，故而计数频率是 100 Hz。整个炖煲的工作时间是由这个计数频率为时钟信号计算的。每计 100 个数就产生一秒信号，然后形成分钟、小时信号。

主程序主要由初态判断、时间预置、工作控制三大部分组成。

初态判断主要是判断是否有水，是否预置时间。当插入电源后，程序自动执行，先把显示器清"0"，接着判断是否有水。无水状态是不允许工作的。故一旦判断出无水则马上报警，并且停止往下执行，等待用户加水，以确保安全。如果显示器 L_1,L_2 都为 0，表明未预置任何时间，这时按下启动按钮 N_1 没有意义，故而也不执行启动工作。

时间预置程序用于设定延时时间和工作时间。按钮 N_2 用于设定延时时间，它是从0 h到 9 h以加法方式设定的。当 L_1 显示为 9 时，按 N_2 则会恢复为 0，即实现循环加法设定。延时时间用于控制用户启动炖煲到实际进行加热蒸炖这段时间。按 N_3 以循环加法方式设定工作时间(加热丝加热时间)，工作时间从 0 h～5 h 循环设定。

工作控制程序段是在预置时间已完成，并按下启动按钮 N_1 时开始的。在按下 N_1 时，炖煲进入了预置时间倒计时的过程。在此之前先判断是否有水，如果无水，则报警，并返回到非启动状态，等待加水和重新启动。如果有水，执行预置延时时间倒计时。当 L_1 倒计时为 0 时，进入加热工作过程。这时，工作时间执行倒计时，同时判断是否沸腾，如果沸腾则进入减功率控制，保

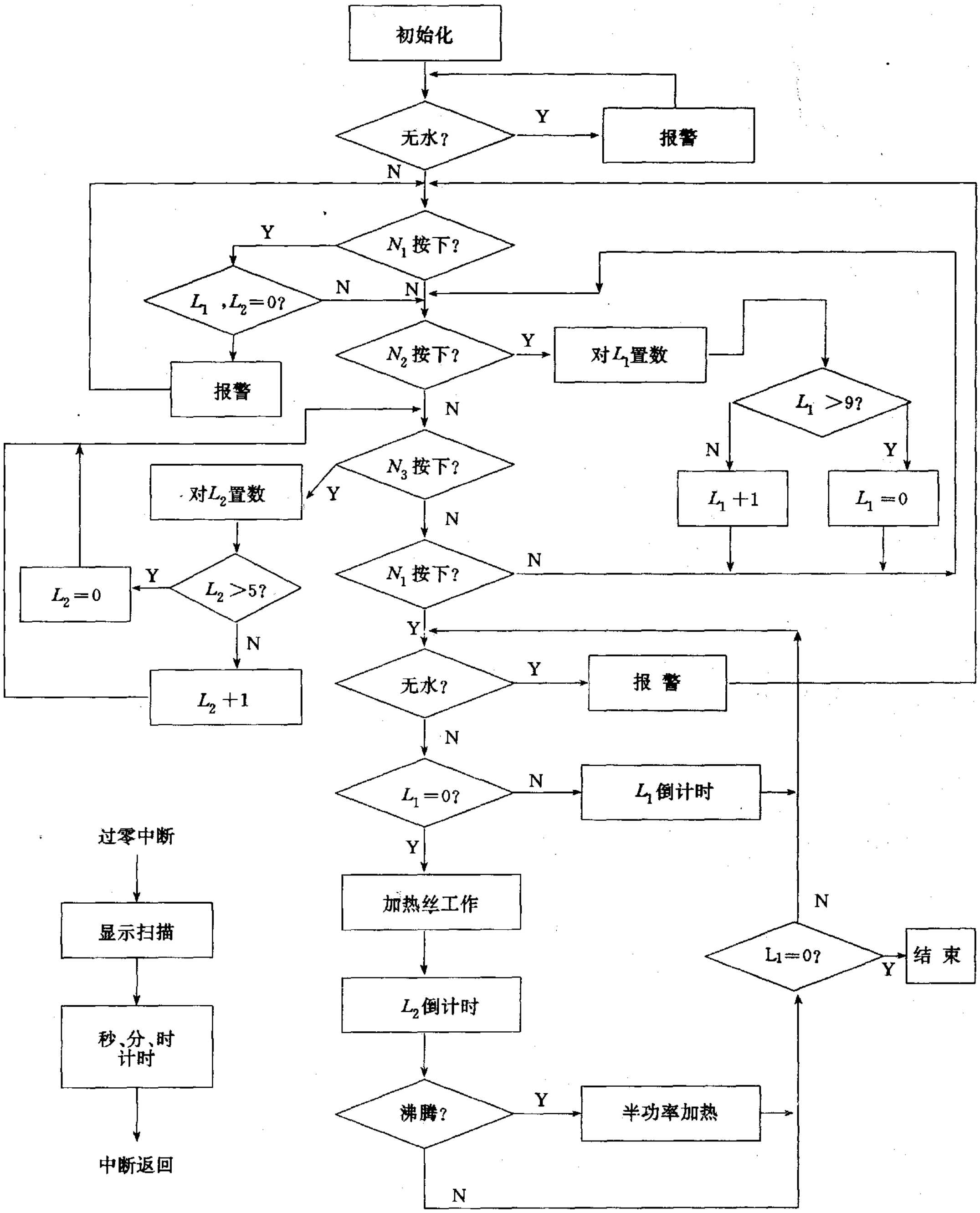


图6-9 控制软件框图

持足够的蒸汽而又不浪费电能。如果 L_2 倒计时未到 0，则继续循环执行工作控制程序段。如果 L_2 为 0，说明工作时间结束，应结束加热。这时，停止加热并向用户报警，提示用户炖煲的工作过程已完成。

蒸炖煲控制系统在工作过程中会显示或者发音报告有关信息。在操作按键时也会显示和发音表示有关信息，这些操作和状态信息如下：

(1) 当用户插上电源插头按通电源时，系统执行初始化，显示“00”；同时，蜂鸣器会发出一声响声，表示已接通电源，等待输入。

(2) 如果这时炖煲内无水，则蜂鸣器会连续鸣叫进行报警，并且停止往下执行。这时，如果用户加水后按动 N_1 按钮，则蜂鸣一声，显示器显示“00”，等待用户设定预置时间和工作时间。

(3) 按 N_2 一次，则显示器 L_1 加一个数，数字由 0～9 循环变化，这时设定预置时间。按 N_3 一次，显示器 L_2 加一个数，数字由 0～5 循环变化，这时设定工作时间。无论 N_2 或 N_3，如果按住不放，则每 0.5 s 加一个数。

(4) 在预置时间、工作时间设定后，按 N_1 则系统进入工作过程。如果预置时间和工作时间为“00”，按 N_1 后，蜂鸣器鸣叫十一声回到步骤(1)。

(5) 若 L_1 显示的预置时间不是零，在按下 N_1 后，L_1 所示的数字以 1 秒亮、1 秒熄的方式闪烁，表示进行预置时间开始倒时计数。每一个小时，显示内容自动减 1。当 L_1 减为 0，且 L_2 显示的工作时间不为零时，则进入加热工作状态。

(6) 若 L_1 显示的预置时间为零，且 L_2 显示的工作时间不为零，按 N_1 则进入加热工作状态。

(7) 进入加热工作状态时，工作指示发光二极管亮，同时 L_2 显示的数字以 1 秒亮、1 秒熄的方式闪烁，进行工作时间倒计时。

(8) 在加热工作过程中，如果炖煲的水被加热蒸发而干水，则蜂鸣器会连续鸣叫报警，显示器全熄，等待用户加水后继续按原状态进行工作。如果水被加热到沸腾，则进入减功率控制。干水时按 N_1 则返回步骤(1)。

(9) 加热过程在 L_2 显示的数字倒计数为 0 时正式结束。这时，蜂鸣器鸣叫十一声后返回步骤(1)。

6.2 68HC05K0 构成的遥控器

MC68HC05K0 是一种价格低、引脚少的单片机。它有 504 字节用户 ROM 和 32 字节 RAM。MC68HC05K0 是 68HC05K 系列单片机中的一员，为 16 只引脚的 DIL 或 SOIC 封装；采用与其他 68HC05 系列单片机一样的 CPU，并具有同样的指令和寄存器；另外，还有一个 15 级的多功能定时器和 10 条通用的双向 I/O 线；掩膜选择允许以软件编程选择所有 I/O 引脚的下拉功能，其中 4 只引脚还可以产生中断。

由于端口引脚具有下拉功能和中断电路，所以 MC68HC05K0 非常适合于做遥控器的键盘。用 MC68HC05K0 做键盘时，除需要一些按键以外不再需要任何其他外部元件。它不再需要外部的上拉或下拉电阻及用作线“或”中断的二极管，因为这些性能都已经设计在该芯片内了。

MC68HC05K0 的这种用途利用了芯片的许多性能来进行红外电视机遥控，而且只要做些简单的修改就可控制其他具有类似传输协议的装置，并且可在 K 系列单片机中运行而无需改动。

6.2.1 遥控器的基本原理

电视机遥控器的基本用途就是向电视机发送控制指令。指令通过敲击遥控器键盘上的按键来产生，键钮按下的检测和解码以及编码的传输则由遥控器中的微控制器来完成。

当遥控器键盘上的一个键钮被按下时，微控制器首先要确定是哪个键被按下并产生该按键的特定代码，然后将按键代码转换为相应的指令代码，该指令代码插到采用规定协议的传输命令中并发送到电视机的接收器。只要按键仍然按下，该命令就不停地发送。

由于遥控器采用电池供电，所以必须使用尽可能小的功率。这可通过让 MCU 在没有键钮被按下时进入 STOP 方式并有效关断电源来实现，当有键钮按下时，产生一个中断请求使 MCU 退出 STOP 方式。

68HC05K0 有 10 个通用 I/O 引脚，其中一只引脚用作传输信号的输出，剩下的 9 只引脚用作键盘控制。剩下的这些引脚中，PA 的 4 只引脚有内部的中断请求电路，用它们来作输入可不用外部中断电路就能检测到键钮的按下，剩余的 5 只引脚留作输出。

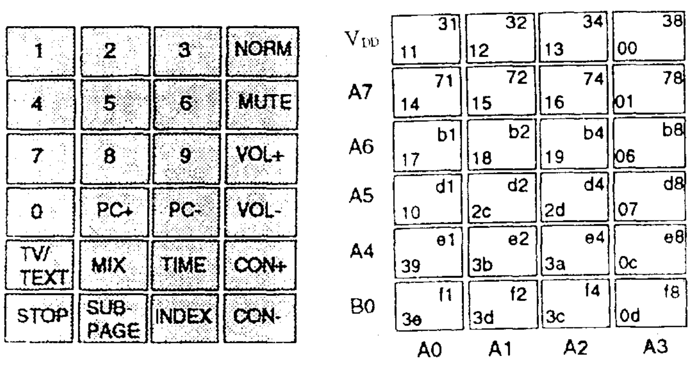
利用内部下拉元件和 4 个输入端上升沿中断请求信号可以产生中断。如果 5 个输出都置“1”，当一个键钮按下时使一个输入由“0”变为“1”，可产生一个中断请求。按这种结构可采用 5×4的键盘。如果用 V_{DD}线来驱动一行 4 个键为“1”，则可另外多控制 4 个键，这样最多可控制 24 个键。

一个键按下后会使其中一列输入变为“1”，对列进行扫描，并逐一使每行输出变为“0”，然后检查是否所有输入都变为“0”，这样就可确定按钮所在的行。如果循环使 5 个输出引脚变为“0 都不能找到按键所在的列，则该键一定是连到 V_{DD}线上的。这种过程给出每个键钮的特定代码，就是列输入和行输出代码的组合。然后，将这个代码译为一条指令插入输出信号中传输出去。

图 6-10(a)是键盘布局，图 6-10(b)是键盘的扫描和传输代码。键盘布局包括了各种电视机控制及 TELE TEXT 的控制。在图 6-10(b)中，键盘的扫描代码见每个按键的右上角，传输代码位于每个按键的左下角；图 6-10(b)还给出了每个按键的行和列的 I/O 引脚。

68HC05K0 构成的遥控器中，信息发送需要一定的传输协议。

这里所说的传输协议是 MC144105 红外遥控发送器所采用的。它采用二进制编码的 9 位数据信息，先发送数据的最低有效位 LSB。每位被发送的信号是一个两相的脉冲编码调制(PCM)信号。PCM 信号的位编码如图 6-11 所示。“0”的位编码是先暂停 512 μs 后跟着 512 μs 的 32 kHz 脉冲串，“1”的位编码则是 32 kHz 的脉冲串后面暂停 512 μs。每位的时间为 1 024 μs。



(a)　　　　(b)

图 6-10　键盘外观及其扫描传输码

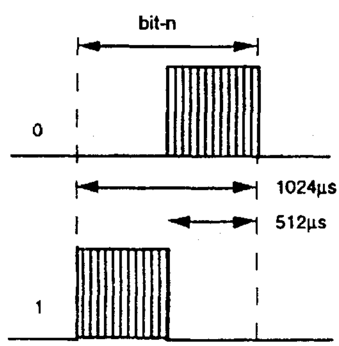


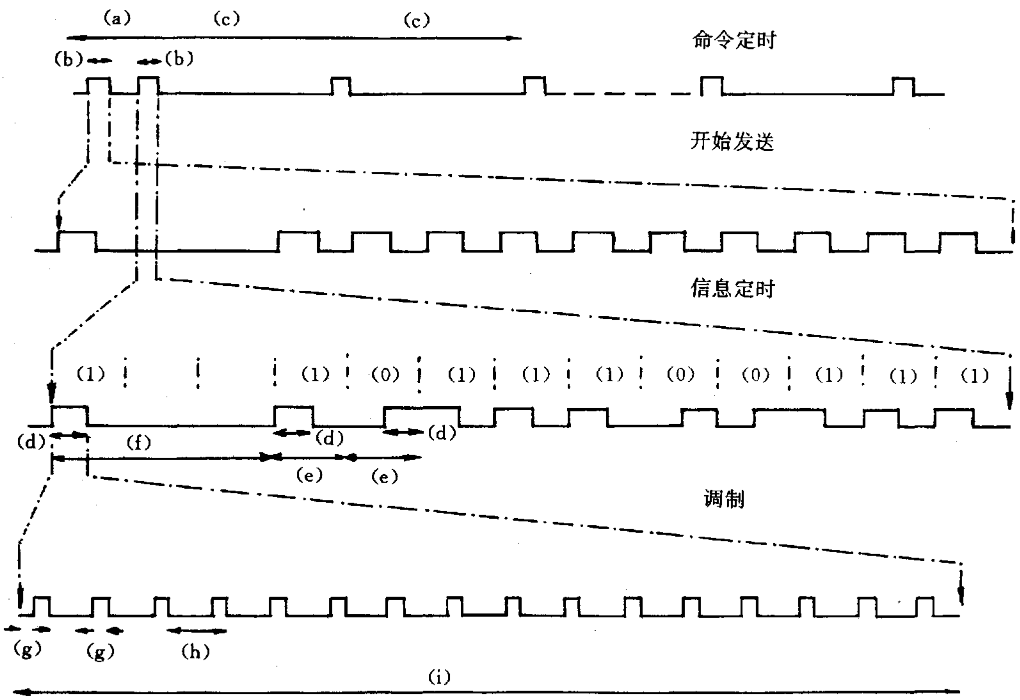
图 6-11　PCM 信号的位编码

一条完整的传输命令由几个信息段组成。每条指令开始是由 9 个“1”组成的起始信息，接着是所按键的相应信息，这个信息不断重复发送直到按键释放为止。按键释放后再由 9 个“1”组成的结束信息结束指令传输。

每个信息段都由 1 个预备位、预备位的间歇、1 个起始位和 9 个数据位组成。预备位和起始位总为“1”。预备位便于调整接收前置放大器的自动增益控制。图 6-12 是传输的时序图。

由图 6-12 可知，起始信息发送完后，按键信息每隔 131 ms(大约 8 Hz)不断重复发送直到按键释放为止，如图中时间(c)所示。9 位指令 111001110 由 LSB 开始发送。预备位的暂停时间等于两个位周期，后面接着是起始位(逻辑“1”)。在发送“0”和“1”的转换期间，脉冲串是连续的，调制脉冲串的频率大约为 32 kHz，占空比为 1∶3。

传输的信号由一个端口引脚输出，用来驱动一个红外二极管放大电路。



命令定时	(a)=32.8ms (b)=13.3ms (c)=131ms	开始发送 开始命令 控制发送	13・位时间
信息定时	(d)=512μs (e)=1.024ms (f)=3.072ms	半位时间 位时间 预脉冲时间	16/f载波
调制	(g)=8μs (h)=32μs (i)=512μs	(1/f载波) (半位时间)	16f载波

图 6-12　传输信息的时序图

6.2.2 遥控器的工作过程

图 6-13 是遥控器在接通电源或复位时的工作流程图。遥控器设置好端口的初始状态为输入或输出后，进入 STOP 方式。只要遥控器没有被复位或没有键被按下，则一直处于 STOP 方式。当有键按下时，产生一个中断请求。短时间的延迟用来确定是真的有键按下而并非噪声干扰。

同时，允许在检测输入之前在输入端有一定的开关作用时间。然后读键盘，确定是哪个键按下，将该键的代码译为一条指令，并传送给电视机。如果按钮一直被按下，则重复发送指令直到按键被释放为止，这对选择电视机频道、调节其音量、颜色和亮度等控制指令是非常有用的。

当按键释放后，向接收器发送一条结束指令，通知它要接收的下一个信息是另一条指令。这对发送像频道数这样的一次性指令是非常有用的。在这个例子中，接收器一次就可调到某个频道。如果要调到另一个频道，则必须将按键释放后接收新的指令序列。

传输结束后端口复位，等待下一个键按下，遥控器回到 STOP 方式。

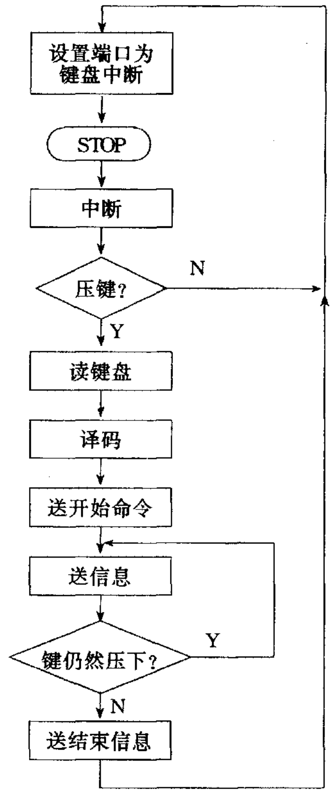


图6-13 工作流程

遥控器的硬件部分由 68HC05K0 芯片、键盘、振荡器和红外放大器组成，原理电路如图 6-14 所示。振荡器可以是石英晶体振荡器或陶瓷振荡器，振荡频率为 2 MHz。由于传输定时基于 1 MHz 的内部时钟频率，所以振荡器的频率是非常重要的。红外放大器采用 2 个晶体管和 2 个普通二极管，限制流过红外二极管的电流大约为 1 A 左右。由于电路中存在较大的开关电流，所以在靠近红外二极管处要接一个大电容。

遥控器软件由几部分组成。

程序的第一部分 MCU 进入 STOP 方式之前设置端口的状态，然后等待键按下。PA0～PA3 设置为带下拉功能的输入端；PA4～PA7 以及 PB0 设置为“1”输出；PB1 设置为“0”输出，使红外放大器在进入 STOP 方式之前不工作。

程序的第二部分称为 PRESD，是一段由中断向量指向的程序，当键按下时会进入该程序。在这段程序中，首先调用键盘扫描子程序确定是哪个键被按下，接着调用译码子程序，将键盘的代码转换为电视机能接收的代码；然后发送起始信息以及指令信息；再检查看是否同一键仍被按下，如果是则重复发送指令信息直到按键释放；最后发送结束信息。

由于传输协议要求有 9 个数据位，而译码后仅为单字节指令，所以要给传输指令的第 9 位设置一个标志。开始和结束传输时该标志置为“1”，产生 9 个“1”的信息，对所有的指令因其第 9 位为“0”，所以置该标志为“0”。

译码子程序将键盘扫描子程序得到的代码与数据矩阵 KEYDAT 进行比较，当它们相同

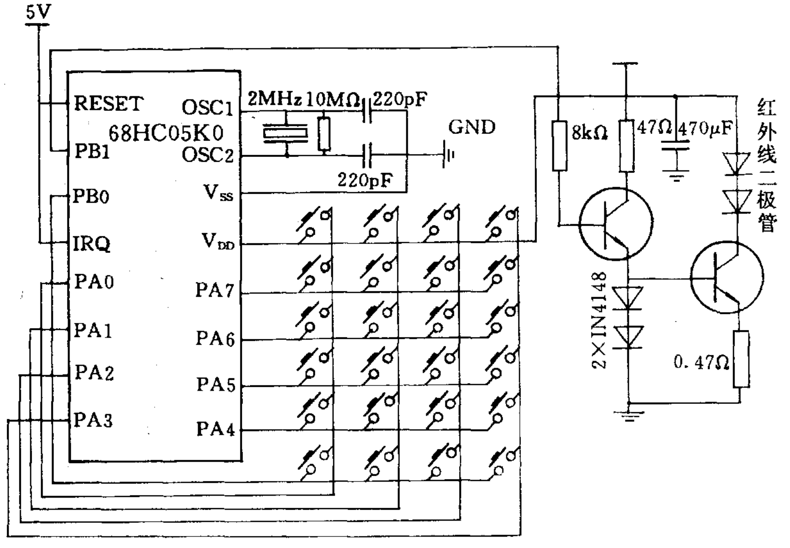


图 6-14 遥控器电路原理图

时，从矩阵 TYDAT 中取出相应的元素作为传输的指令代码。

图 6-10(b)中左下角给出的指令代码是专门用于接收器的。每一个采用同样的通信协议的接收器都接收同样的 9 位指令，但指令代码是什么就取决于接收器的软件。在本例中，8 位指令“14”表示选择第四频道；在另外一个接收器中，指令代码“14”可能表示增加音量。

“bsr trnmit”指令是转子指令，它使程序转向 trnmit 子程序。程序进入传送子程序 trnmit 时传送 KEYST3 的内容。在传送了预备位和开始位之后，指令字节从最低有效位 LSB 开始循环送入进位位。如果进位标志为“1”，则传送逻辑“1”；如果进位标志为“0”，则传送逻辑“0”。每一位的发送如图 6-11 所示。子程序 SEND0 发送 512 μs 的 32 kHz 脉冲串，子程序 SEND1 发送 512 μs 的 32 kHz 脉冲串之后暂停 512 μs。在发送“0”之后接着发送“1”，则会连续发送 1 024 μs 的脉冲串。为了避免在这个脉冲串中出现断结，在 SEND0 子程序中对下次发送位进行检测，以确定是否发送双长度脉冲串。随后，SEND1 子程序必须检测确定在上一个“1”和发送脉冲串前的半位周期中并没有发送双长度脉冲串。

子程序 BURST 按累加器的内容产生相应持续时间的 32 kHz 脉冲串。当对 B 端口的 PB1 设置高或低电平的指令时间是 5 个时钟周期时，最小处理机时钟周期是通过下面方法得到的，即用改变 PB1 输出状态的最小时钟周期数除最小输出状态时间。在输出为高电平时，最小输出状态时间是 8 μs。这样，内部时钟周期就是 8 μs/5＝1.6 μs。对改变 PB1 输出状态的指令时间多加 3 个周期延时，则成为 8 个时钟周期，这样，所需要的内部时钟周期为 8μs/8＝1 μs。所以，允许用 2 MHz 的振荡器。

程序代码的大小大约是 300 字节，留下的存储器空间可以用于增加遥控器的功能。

在运行中可以首先调试程序，以保证其工作正确。

在加上电源时，电路产生复位，软件的首部会对程序计数器初始化。当用示波器或逻辑分析仪检查 PB1 的输出时必须注意：当试图捕捉按下键的信号时，第一个信号输出将是 9 个“1”的起始信息。为了捕捉指令，一直压住按键，这样指令将连续反复发送，随后捕捉可以在这点开

始，从而可知发出信息的传输码。

6.2.3 遥控器的控制软件清单

下面给出了遥控器详细的控制软件清单。控制软件主要包括 8 个程序段：第一个程序段 start 对 68HC05K0 进行初始化，以适应遥控器电路工作情况；第二个程序段 presd 是主程序，它对键盘进行检测译码；第三个程序段 keycn 对键盘进行扫描并寻键；第四个程序段 decode 对键盘进行译码；第五个程序段 trnmit 进行发送处理；第六个程序段 send1 形成 PCM 格式的“1”；第七个程序段 send0 形成 PCM 格式的“0”；第八个程序段 burst 形成脉冲发送所需的标志。

在软件清单中，每个程序段以及相应的指令都给出了注释。所以，每一程序段的详细功能以及具体意义可以通过对具体指令的理解而取得，并且在注释的提示下明确其实质的工作过程及作用。详细的软件清单如下：

```
          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          * INFAR RED REMOTE CONTROL FOR K0, K1                                    *
          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          * WRITTEN BY A.BRESLIN    13/1/92                                        *
          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          * THIS PROGRAM READS AND ENCODES A KEY FROM A 24 KEY                     *
          * KEYBOARD TO A FORM OF BIPHASE PULSE CODE MODULATION                   *
          * (FCM) FOR INFRA RED TRANSMISSION. IT USES THE TRANSMISSION            *
          *  PROTOCOL OF THE MC144105 IR REMOTE CONTROL                           *
          * TRANSMITTER                                                           *
          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


0000      porta    equ   00
0001      portb    equ   01
0004      ddra     equ   04
0005      ddrb     equ   05
0008      tcsr     equ   $08
0010      papd     equ   $10

00e0               org   $e0

00e0      keyst1   rmb   1                 ;initial code form keyboard
00e1      keyst2   rmb   1                 ;keycode
00e2      keyst3   rmb   1                 ;code transmitted
00e3      dflag    rmb   1                 ;flag for last and 9th bits

```

```
               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               * THE PORTS ARE SET UP USING PORTA 0-3 AS INPUTS MAKING                   *
               * USE OF THE INTERNAL INTERRUPT GENERATION ON THESE                       *
               * I/O LINES.  STOP MODE IS ENTERED UNTIL A KEY IS PRESSED                 *
               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0200                    org     $ 200

0020 9a     start       cli
0201 ad 04  wpres       bsr     setup
0203 9c                 rsp
0204 8e                 stop
0205 20 fa              bra     wpres

0207 a6 f0  setup       lda     # $ f0          ;porta 0~3 inputs
0209 b7 04              sta     ddra            ;4~7 as outputs
020b b7 00              sta     porta           ;set outputs high
020d b7 10              sta     papd            0~3 pulldown
020f a6 03              lda     # $ 03          ;portb 0~1 outputs
0211 b7 05              sta     ddrb
0213 a6 01              lda     # $ 01          ;set portb 0 high
0215 b7 01              sta     portb
0217 81                 rts


               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               * THE KEY READ IS DECODED FOR TRANSMISSION.                               *
               * THE TRANSMISSION PROTOCOL REQUIRES A START MESSAGE OF 9                 *
               * ONES FOLLOWED BY THE KEYPRESSED CODE.  THIS CODE IS                     *
               * CONTINUALLY RETRANSMITTED IF THE KEY IS HELD DOWN.                      *
               *  AN END CODE OF 9 ONES TERMINATES THE TRANSMISSION                      *
               * AND THE DEVICE RETURNS TO STOP MODE.                                    *
               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0218 ad 34   presd      bsr     keyscn          ;get key pressed
021a b6 e1              lda     keyst2          ;save key to check
021c b7 e0              sta     keyst1          ;if key held down
021e ad 67              bsr     decode          ;decode key pressed
0220 12 e3              bset    1,dflag         ;set nineth bit to 1
0222 a6 ff              lda     # $ ff          ;send start data
0224 b7 e2              sta     keyst3          ;to transmission routine
0226 ad 71              bsr     trnmit          ;nine ones
0228 b6 e1   sndagn     lda     keyst2          ;send key press message byte
022a b7 e2              sta     keyst3
```

```
022c 13 e3               bclr    1,dflag         ;set nineth bit to 0
022e ad 69               bsr     trnmit
0230 b6 00               lda     porta           ;check if key still pressed
0232 a4 0f               and     # $ 0f          ;end if no key pressed
0234 26 0f               bne     endtrn
0236 ad 16               bsr     keyscn          ;else check if same
0238 b6 e0               lda     keyst1          ;key pressed
023a b1 e1               cmp     keyst2
023c 26 07               bne     endtrn          end if not
023e ae c8               ldx     # $ c8          ;delay
0240 5a         tloop    decx                    ;before next
0241 26 fd               bne     tloop           transmission
0243 20 e3               bra     sndagn
0245 12 e3      endtrn   bset    1,dflag         ;send end message
0247 a6 ff               lda     # $ ff          ;of nine ones
0249 b7 e2               sta     keyst3
024b ad 4c               bsr     trnmit
024d 80                  rti                     ;re-enter stop mode

                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                * WHEN A KEY IS PRESSED THE DEVICE COMES OUT OF STOP MODE     *
                * THE KEYBOARD IS SCANNED TO SEE WHICH KEY IS PRESSED         *
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

024e cd 02 fc   keyscn   jsr     datwt           ;wait for debounce
0251 b6 00               lda     porta           ;check if key press
0253 b7 e0               sta     keyst1          ;store inputs
0255 a4 0f               and     # $ 0f          ;mask outputs
0257 27 a7               beq     start           ;stop if no key pressed
0259 ae ef               ldx     # $ ef          ;set one row low
025b 9f         nxtrow   txa                     ;read ouput lines
025c b4 e0               and     keyst1          ;combine with inputs
025e b7 e1               sta     keyst2          ;store key code
0260 bf 00               stx     porta           ;to find row which clears inputs
0262 b6 00               lda     porta           ;check for inputs cleared
0264 a4 0f               and     # $ 0f          ;mask outputs
0266 27 1c               beq     gotit           ;zero in key-press row clears inputs
0268 58                  lslx                    ;check if last row
0269 5c                  incx                    ;set lsb to 1
026a 24 02               bcc     tryb            ;try portb output if not porta
026c 20 ed               bra     nxtrow          ;try next porta output row

026e b6 e0      tryb     lda     keyst1
0270 b7 e1               sta     keyst2
0272 ae f0               ldx     # $ f0
```

```
0274 bf 00              stx     porta           ;sedt all porta outputs high
0276 11 01              bclr    0,portb         ;set portb 0 output low
0278 b6 00              lda     porta           ;check for inputs cleared
027a a4 0f              and     #$0f            ;mask outputs
027c 27 06              beq     gotit           ;zero in key-press row clears inputs
027e b6 e1              lda     keyst2          ;
0280 a4 3f              and     #$3f            ;set individual code since last row
0282 b7 e1              sta     keyst2          ;store code
0284 10 01      gotit   bset    0,portb         ;set portb column high again
0286 81                 rts

        ****************************************************************
        * THE DECODE ROUTINE USES TWO ARRAYS, IT COMPARES THE KEY      *
        * VALUE WITH THE ARRAY KEYDAT AND WHEN A MATCH IS FOUND THE    *
        * CORRESPONDING ELEMENT IN THE ARRAY TVDAT BECOMES THE         *
        * TRANSMITIED CODE.                                            *
        ****************************************************************

0287 ae 18      decode  ldx     #$18            ;data array offset to zero
0289 d6 03 02   nxtel   lda     keydat.x        ;look at each element of array
028c b1 e1              cmp     keyst2          ;compare with key read
028e 27 03              beq     match           ;decode if match
0290 5a                 decx                    ;else try next element
0291 26 f6              bne     nxtel           ;norm if no match found
0293 d6 03 1a   match   lda     tvdat,x         ;get key code
0296 b7 e1              sta     keyst2          ;store code to transmit
0298 81         rts

                ****************************************************************
                * THE TRANSMISSION PROTOCOL REQUIRES A PRE-BIT,A PRE-BIT       *
                * PAUSE, A START BIT AND NINE DATA BITS, WHERE THE PRE-BIT     *
                * AND THE START BIT ARE LOGIC '1'.                             *
                ****************************************************************

0299 10 e3      trnmit  bset    0,dflag         ;initialize for first bit
029b ad 32              bsr     send1           ;send pre-bit
029d cd 02 fc           jsr     datwt           ;pre-bit pause
02a0 cd 02 fc           jsr     datwt           ;equalling four half data periods
02a3 cd 02 fc           jsr     datwt           ;
02a6 cd 02 fc           jsr     datwt           ;
02a9 ad 24              bsr     send1           ;send start bit
02ab ae 08              ldx     #$08            ;transmit 8 data bits
02ad 34 e2      nxtbit  lsr     keyst3          ;get next bit
02af 25 04              bcs     data1           ;send 1 if carry set
02b1 ad 28              bsr     send0           ;send 0 if carry clear
```

```
02b3 20 02                  bra     bitsnt
02b5 ad 18       data1      bsr     send1
02b7 5a          bitsnt     decx                        ;countdown bits sent
02b8 26 f3                  bne     nxtbit              ;send next bit if count not zero
02ba 03 e3 04               brclr   1,dflag,send00      ;if flag set
02bd ad 10                  bsr     send1               ;send 1 as nineth bit
02bf 20 02                  bra     endend              ;
02c1 ad 18       send00     bsr     send0               ;else send 0
02c3 ae 18       endend     ldx     # $18
02c5 ad 35       loopw      bsr     datwt               ;delay between successive
02c7 ad 33                  bsr     datwt               ;transmissions
02c9 ad 31                  bsr     datwt
02cb 5a                     decx
02cc 26 f7                  bne     loopw
02ce 81                     rts


                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 * TO TRANSMIT A LOGIC '1' A 32kHz PULSE TRAIN FOR 512 μs IS        *
                 * FOLLOWED BY A 512 μs PAUSE.                                      *
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

02cf 01 e3 04    send1      brclr   0,dflag,last0       ;check if last bit was zero
02d2 a6 10                  lda     # $10               ;burst if last bit was 1
02d4 ad 15                  bsr     burst               ;32kHz pulse for 512 μs
02d6 ad 24       last0      bsr     datwt               ;wait 512 μs
02d8 10 e3                  bset    0,dflag             ;set flag as 1 sent
02da 81                     rts

                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 * TO TRANSMIT A LOGIC '0' A 512 μs PAUSE IS FOLLOWED BY A          *
                 * 32 kHz PULSE TRAIN FOR 512 μs.  IF A LOGIC '1' FOLLOWS A '0'     *
                 * THE 32 kHz IS CONTINUED FOR 1024 μs TO AVOID A PROCESSING        *
                 * DELAY                                                            *
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

02db ad 1f       send0      bst     datwt               ;wait 512 μs
02dd 00 e2 04               brset   0,keyst3, next1     ;check if next bit is 1
02e0 a6 10                  lda     # $10               ;single burst if 1
02e2 20 02                  bra     datset              ;data set.
02e4 a6 20       next1      lda     # $20               ;double burst required
02e6 ad 03       datset     bsr     burst               ;32 kHz pulse for 512 μs
02e8 11 e3                  bclr    0,dflag             ;clear flag as 0 sent
02ea 81                     rts

```

```
               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               * THE 32 kHz PULSE TRAIN HAS A MARK TO SPACE RATIO OF 1 TO 3 *
               * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

02eb 13 01         burst     bclr   1,portb            ;portb 1 low
02ed 21 fe                   brn    *
02ef 12 01                   bset   1,portb            ;portb 1 high
02f1 21 fe                   brn    *
02f3 13 01                   bclr   1, portb           ;portb 1 low
02f5 9d                      nop
02f6 4a                      deca                      ;decrement count
02f7 27 02                   deq    endbur             ;end of burst ?
02f9 20 f0                   bra    burst
02fb 81            endbur    rts


02fc a6 52         datwt     lda    # $52              ;count
02fc 4a            loop      deca                      ;to provide 512 μs delay
02ff 26 fd                   bne    loop               ;after instruction times
0301 81                      rts

0302 31 f1 e1 d1 b1 71   keydat   fcb   $31, $f1, $e1, $d1, $b1, $71
0308 32 f2 e2 d2 b2 72            fcb   $32, $f2, $e2, $d2, $b2, $72
030e 34 f4 e4 d4 b4 74            fcb   $34, $f4, $e4, $d4, $b4, $74
0314 38 f8 e8 d8 b8 78            fcb   $38, $f8, $e8, $d8, $b8, $78

031a 11 3e 39 10 17 14   tvdat    fcb   $11, $3e, $39, $10, $17, $14
0320 12 3d 3b 2c 18 15            fcb   $12, $3d, $3b, $2c, $18, $15
0326 13 3c 3a 2d 19 16            fcb   $13, $3c, $3a, $2d, $19, $16
032c 00 0d 0c 07 06 01            fcb   $00, $0d, $0c, $07, $06, $01


0332 80                  softin   rti

03fa                  org   $3fa

03fa 02 18            fdb   presd           ;scan keybrd on int
03fc 03 32            fdb   softin          ;software interrupt
03fe 02 00            fdb   start           ;reset
```

6.3 68HC05K0 控制的温度检测仪

温度检测仪是一个可以检测和显示温度的电子仪表。它在塑料加工、药品加工和工业控制中有很多用途。用 68HC05K0 可以构成一个温度检测仪，在此基础上稍作修改就可以变成一个温度检测控制仪。这在实际生产中有更大的用场。

在温度检测中，要把模拟量的温度值送入 68HC05K0 中就需要有 A/D 转换器。而 68HC05K0 中片内并没有 A/D 转换部件。因此，需要采用一种较合理的方法进行 A/D 转换，同时，这种方法必须有廉价易行的特点。

在这一节中，将介绍 68HC05K0 构成温度检测仪的有关原理，A/D 转换的方法，硬件电路及其作用，软件框图及其工作原理。

6.3.1 温度检测仪的基本原理

温度检测仪由电源、68HC05K0 单片机、显示电路、A/D 转换电路等组成。温度检测仪的结构框图如图 6-15 所示。从图中可以看出，单片机 68HC05K0 是温度检测仪的核心。

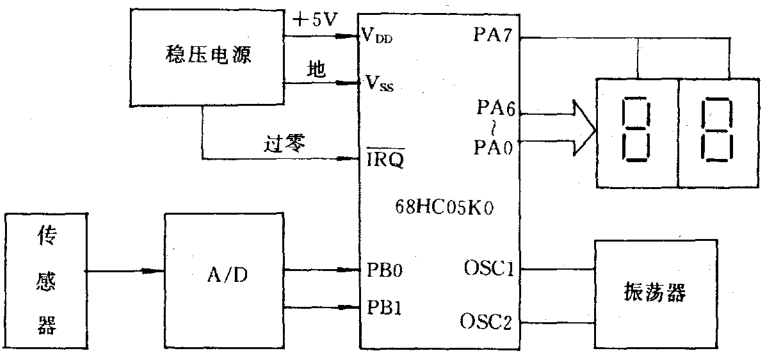


图 6-15 温度检测仪结构框图

稳压电源把 220 V 的交流电压进行降压，然后进行整流处理，通过稳压电路之后产生稳定的＋5 V 直流电压，再向单片机 68HC05K0 供电。68HC05K0 有了＋5 V 电源供电之后就可以正常工作了。

振荡器是外部振荡元件，它和 68HC05K0 内部的振荡器相连组成一个完整的振荡器电路，产生 68HC05K0 工作时所需的脉冲时钟。单片机 68HC05K0 的外部振荡元件可以用石英晶体、陶瓷振荡器、二端电阻电容、三端电阻电容等。为了经济起见，温度检测仪可以采用二端电阻电容组成振荡器。

传感器是用半导体热敏电阻作传感元件的，由于半导体热敏电阻有热传感灵敏度高的特点，所以可以得到较准确的温度值，也即是温度检测仪有较好的分辨率。半导体热敏电阻本质上是采用半导体 PN 结的热敏效应做成的。半导体 PN 结在温度越高时漏电流越大，表现为电阻越小；在温度越低时漏电流越小，电阻越大。这种特点也称为负温度特性。典型的半导体热敏电阻的特性如图 6-16 所示。

图 6-16 中给出了 9 条半导体热敏电阻的特性曲线。这 9 条曲线分别用数字 1～9 标识出

来。在 25 ℃时，它们的电阻值分别是 1 kΩ，2 kΩ，3 kΩ，5 kΩ，10 kΩ，20 kΩ，30 kΩ，50 kΩ，100 kΩ。

半导体热敏电阻有 6 个十分重要的参数，这些参数用于描述半导体热敏电阻的有关性能。下面分别介绍这 6 个参数的名称和意义。

1. 零功率电阻值 R_T

在规定的温度下，所用的测量功率能引起的电阻变化相对于总的测量误差来说可以忽略不计，这时测出的电阻值称为零功率电阻值 R_T。

2. 额定零功率电阻值 R_{25}

热敏电阻的设计电阻值，通常是指在 25 ℃时测得的零功率电阻值。这个电阻一般标在热敏电阻上，称为额定零功率电阻值，以 R_{25}表示。

3. B 值

半导体热敏电阻有高的负温度系数，电阻和温度之间的关系可以用下面的方程表示：

$$R_{T_1} = R_{T_0}\exp[B(1/T_1 - 1/T_0)] \qquad (6\text{-}2)$$

其中，R_{T_0}是热敏电阻在绝对温度 T_0 时的电阻值；

R_{T_1}是热敏电阻在绝对温度 T_1 时的电阻值；

B 是给定的热敏电阻的特性常数。

式(6-2)中的 B 常数也称为 B 值，它是热敏电阻在制造时所用材料的函数。B 值可以通过测量热敏电阻在 T_1，T_0 两个温度点的电阻值来确定。

$$B = \frac{\ln R_{T_1} - \ln R_{T_0}}{1/T_1 - 1/T_0} \qquad (6\text{-}3)$$

即

$$B = \ln \frac{R_{T_1}}{R_{T_0}} \Big/ (1/T_1 - 1/T_0)$$

$$= \frac{T_1 T_0}{T_0 - T_1} \ln \frac{R_{T_1}}{R_{T_0}} \qquad (6\text{-}4)$$

其中，T_0 是绝对温度的一个温度点；

T_1 是绝对温度的一个温度点；

R_{T_0}是 T_0 时的零功率电阻值；

R_{T_1}是 T_1 时的零功率电阻值。

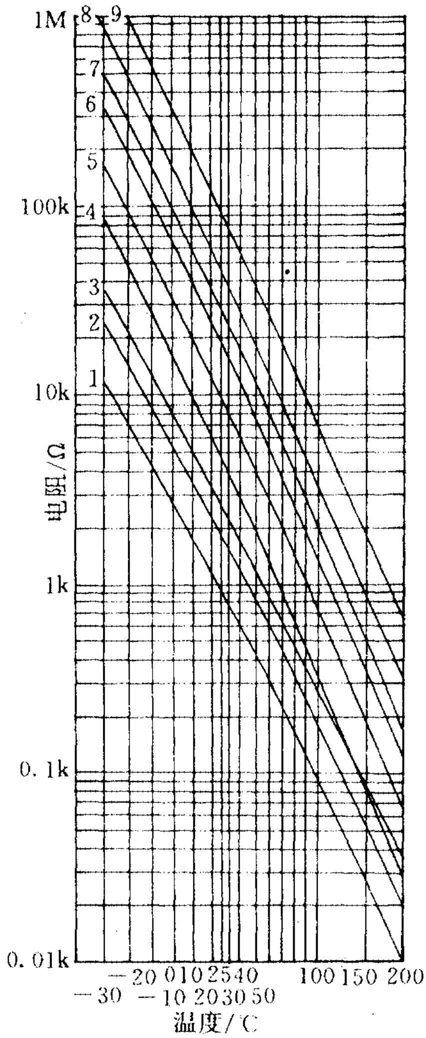


图 6-16 典型的半导体热敏电阻特性

除非特别指明，一般 B 值是由 25 ℃(即 298.15 K)和 85 ℃(即 358.15 K)这两点温度的零功率电阻值计算得出的。

应该指出，B 值在热敏电阻的工作温度范围内并不是一个严格的常数。

4. 零功率电阻温度系数 α

在规定的温度中，热敏电阻的零功率电阻值随温度的变化率与其零功率电阻值之比，称为

零功率电阻温度系数 α。

α 的意义由下面的方程给出：

$$\alpha_T = \frac{1}{R_T} \cdot \frac{\mathrm{d} R_T}{\mathrm{d}T} = -\frac{B}{T^2} \times 100\% \tag{6-5}$$

其中，α_T 是在绝对温度 T 时的零功率电阻温度系数；

R_T 是在绝对温度 T 时的零功率电阻；

T 是绝对温度值(K)；

B 是 B 值。

5. 耗散系数 δ

加在热敏电阻上的功率和热敏电阻由此加热所引起的温度变化之间的关系，称为耗散系数，用 δ 表示。

$$\delta = \frac{P}{T_1 - T_0} = \frac{V \times I}{T_1 - T_0} \tag{6-6}$$

其中，T_0 是原来温度(K)；

T_1 是加了功率后上升到的温度(K)；

P 是所加功率，它是电压 V 与电流 I 的乘积。

在工作温度范围中，δ 值随环境温度变化而有所变化。δ 一般用 mW/C°表示。

6. 热时间常数 τ

在零功率条件下，当温度发生突然变化时，热敏电阻的温度变化为开始到结终两个温度差的 63.2 %所需的时间，称为热时间常数 τ。

τ 和热敏电阻的热容量 C 成正比，与耗散系数 δ 成反比。即

$$\tau = \frac{C}{\delta} \tag{6-7}$$

一般而言，热敏电阻在实际应用中不是用式(6-2)求取电阻值的，而是利用实际热敏电阻的分度表。因为分度表给出了电阻值和温度值的对应值，在实际应用中只需简单查表就可以了。

下面给出 MF58 型热敏电阻的阻值—温度分度表，如表 6-1 中所示。表中所示的热敏电阻 R_{25}=100 kΩ，B(25 ℃/85 ℃)=3 990 K。

A/D 转换器是由 RC 电路组成的电阻—时间(即 R—T)转换器，它可以把不同的电阻值转换成对应的时间周期。

单片机 68HC05K0 根据 A/D 转换的时间可以判断出所检测的电阻值以及相应的温度值。单片机还会把这个温度值送到显示器去显示。在显示时，单片机采用扫描方式，在 PA0～PA6 送出显示码，在 PA7 送出显示位选择码。单片机在扫描时利用外部过零信号产生中断，通过中断程序执行扫描工作。由于每 10 ms 中断一次，故每位显示码在一秒之内可以显示 50 次，从而有较好的亮度和稳定性。

显示器由 2 个 7 段发光二极管显示器和有关开关电路组成，故可以显示 2 位温度，即从 0 ℃～99 ℃。这就是说，温度检测仪可以用来检测 100 ℃以内的温度。

表 6-1 MF58 型热敏电阻的阻值—温度分度表(R_{25}=100 kΩ,$B_{25/85}$=3 990 K)

T/℃	R/kΩ	T/℃	R/kΩ	T/℃	R/kΩ	T/℃	R/kΩ
0.0	327.785 9	38.0	57.510 7	76.0	14.204 6	114.0	4.519 9
1.0	311.505 9	39.0	55.208 4	77.0	13.744 5	115.0	4.396 9
2.0	296.129 8	40.0	53.010 9	78.0	13.301 4	116.0	4.277 7
3.0	281.601 5	41.0	50.912 8	79.0	12.874 9	117.0	4.162 1
4.0	267.858 7	42.0	48.909 2	80.0	12.464 1	118.0	4.050 1
5.0	254.882 9	43.0	46.995 4	81.0	12.068 5	119.0	3.941 5
6.0	242.598 9	44.0	45.167 0	82.0	11.687 3	120.0	3.836 2
7.0	230.974 7	45.0	43.419 7	83.0	11.320 1	121.0	3.734 2
8.0	219.970 9	46.0	41.749 5	84.0	10.966 2	122.0	3.635 2
9.0	209.551 1	47.0	40.152 8	85.0	10.625 0	123.0	3.539 1
10.0	199.681 1	48.0	38.625 8	86.0	10.296 1	124.0	3.446 0
11.0	190.329 0	49.0	37.165 3	87.0	9.978 9	125.0	3.355 6
12.0	181.464 8	50.0	35.768 0	88.0	9.673 0	126.0	3.267 9
13.0	173.060 6	51.0	34.430 9	89.0	9.378 0	127.0	3.182 8
14.0	165.090 3	52.0	33.151 0	90.0	9.093 3	128.0	3.100 3
15.0	157.529 3	53.0	31.925 7	91.0	8.818 5	129.0	3.020 1
16.0	150.354 6	54.0	30.752 3	92.0	8.553 3	130.0	2.942 2
17.0	143.544 5	55.0	29.628 5	93.0	8.297 3	131.0	2.866 7
18.0	137.078 9	56.0	28.551 8	94.0	8.050 2	132.0	2.793 3
19.0	130.938 8	57.0	27.520 1	95.0	7.811 5	133.0	2.722 0
20.0	125.106 1	58.0	26.531 3	96.0	7.580 9	134.0	2.652 8
21.0	119.584 1	59.0	25.583 3	97.0	7.358 2	135.0	2.585 6
22.0	114.296 9	60.0	24.674 3	98.0	7.143 0	136.0	2.520 2
23.0	109.289 7	61.0	23.802 5	99.0	6.935 0	137.0	2.456 8
24.0	104.528 5	62.0	22.968 1	100.0	6.734 0	138.0	2.395 1
25.0	100.000 0	63.0	22.163 6	101.0	6.539 7	139.0	2.335 2
26.0	95.691 8	64.0	21.393 4	102.0	6.351 8	140.0	2.276 9
27.0	91.592 3	65.0	20.654 1	103.0	6.170 1	141.0	2.220 3
28.0	87.690 2	66.0	19.944 2	104.0	5.994 3	142.0	2.165 2
29.0	83.975 4	67.0	19.262 5	105.0	5.824 3	143.0	2.111 7
30.0	80.437 8	68.0	18.607 6	106.0	5.659 9	144.0	2.059 7
31.0	77.068 3	69.0	17.978 5	107.0	5.500 7	145.0	2.009 1
32.0	73.858 1	70.0	17.373 9	108.0	5.346 7	146.0	1.959 8
33.0	70.799 0	71.0	16.792 9	109.0	5.197 5	147.0	1.912 0
34.0	67.883 1	72.0	16.234 0	110.0	5.053 2	148.0	1.805 4
35.0	65.103 1	73.0	15.696 8	111.0	4.913 4	149.0	1.820 1
36.0	62.452 0	74.0	15.180 1	112.0	4.778 1	150.0	1.785 9
37.0	59.923 3	75.0	14.683 0	113.0	4.646 9	151.0	1.747 9

续表 6-1

T/℃	R/kΩ	T/℃	R/kΩ	T/℃	R/kΩ
152.0	1.711 0	188.0	0.807 8	224.0	0.402 4
153.0	1.675 1	189.0	0.791 0	225.0	0.395 0
154.0	1.640 2	190.0	0.774 6	226.0	0.387 7
155.0	1.606 2	191.0	0.758 7	227.0	0.380 6
156.0	1.573 1	192.0	0.743 1	228.0	0.373 6
157.0	1.540 8	193.0	0.728 1	229.0	0.366 8
158.0	1.509 3	194.0	0.713 4	230.0	0.360 1
159.0	1.478 5	195.0	0.699 2	231.0	0.353 5
160.0	1.448 4	196.0	0.685 5	232.0	0.347 1
161.0	1.419 0	197.0	0.672 2	233.0	0.340 8
162.0	1.390 3	198.0	0.659 4	234.0	0.334 6
163.0	1.362 1	199.0	0.647 0	235.0	0.328 5
164.0	1.334 5	200.0	0.635 1	236.0	0.322 6
165.0	1.307 5	201.0	0.622 0	237.0	0.316 8
166.0	1.281 0	202.0	0.612 0	238.0	0.311 1
167.0	1.255 0	203.0	0.599 1	239.0	0.305 6
168.0	1.229 4	204.0	0.587 9	240.0	0.300 1
169.0	1.204 4	205.0	0.576 7	241.0	0.294 5
170.0	1.179 8	206.0	0.565 7	242.0	0.289 5
171.0	1.155 6	207.0	0.555 0	243.0	0.284 4
172.0	1.131 9	208.0	0.544 4	244.0	0.279 4
173.0	1.108 6	209.0	0.534 1	245.0	0.274 5
174.0	1.085 7	210.0	0.524 0	246.0	0.269 7
175.0	1.063 3	211.0	0.514 2	247.0	0.265 0
176.0	1.041 2	212.0	0.504 5	248.0	0.260 3
177.0	1.019 5	213.0	0.495 0	249.0	0.255 8
178.0	0.998 3	214.0	0.485 7	250.0	0.251 4
179.0	0.977 4	215.0	0.476 6		
180.0	0.957 0	216.0	0.467 6		
181.0	0.936 9	217.0	0.458 9		
182.0	0.917 2	218.0	0.450 3		
183.0	0.898 0	219.0	0.441 9		
184.0	0.879 1	220.0	0.433 7		
185.0	0.860 7	221.0	0.425 6		
186.0	0.842 6	222.0	0.417 7		
187.0	0.825 0	223.0	0.410 0		

6.3.2 外接 *RC* 实现 A/D 转换的方法

68HC05K0 内部是没有 A/D 转换器的。要进行模拟量检测，就要用到 A/D 转换器。这样，在 68HC05K0 构成的温度检测仪中，就必须采用一种合适、廉价、简单、有效的方法来组成 A/D转换器，以便对温度进行检测。由于 68HC05K0 单片机是一个有各种处理能力的控制器件，利用外部的 *RC* 电路和 68HC05K0 单片机结合，通过程序控制就能实现较理想的 A/D 转换功能。在这里，介绍几种单片机和外部 *RC* 结合形成 A/D 转换的技术方法。这几种方法包括无偏置电源的 *RC* 方法，有偏置电源的 *RC* 方法以及含运放的 *RC* 方法等三种。

下面分别对这三种 A/D 转换方法进行介绍。

一、无偏置电源的 *RC* 实现 A/D 方法

无偏置电源的 *RC* 方法就是在单片机外部的 *RC* 电路中，不加任何外部偏置电源，而是利用单片机的输出电平直接对 *RC* 进行偏置，并在单片机的程序控制下实现 A/D 转换。

无偏置电源的 *RC* 实现 A/D 转换的电路结构如图 6-17 所示。从图中可以看出，电路由单片机 68HC05K0 和电阻 R_s，R_m，R_d 和电容 C 组成。这是一种十分简单的结构，有着极好的经济性，因为只是在外部加上 3 个电阻和 1 个电容而已。

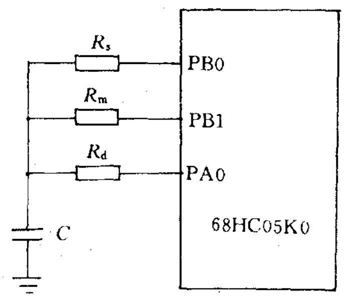


图 6-17　无偏置电源的 *RC* 实现 A/D

在图 6-17 中，R_s 是标准电阻，用于校正和定标；R_m 是被检测的电阻；R_d 是一个限流电阻，用于限制电容 C 过 PA0 端放电的电流值，也即是说，R_d 是一个放电回路的放电电阻。R_d 越小，则放电越快，但放电电流越大。为了使放电电流能保证单片机不会损坏，放电电流常不大于 8 mA，故 R_d 一般取 600 Ω 左右。被检测的电阻 R_m 是个热敏电阻，因此检测出 R_m 的值时就等于检测出了相应的温度。

图 6-17 所示 A/D 转换电路的工作过程如下：

首先假定 PA0 端的输入高电平门槛电压为 V_{IH}，输出的低电平门槛电压为 V_{OL}。在初态时，电容 C 已放电，故而有电压 $V_C=V_{OL}$。下面开始执行 A/D 过程。

1. 定标过程

这时，PA0，PB1 端为输入端口，连接标准电阻 R_s 的 PB0 端输出高电平，此时 PB0 输出的高电平过 R_s 对电容 C 充电。单片机内部定时器开始进行充电计时。计时的时候可采用对定时器溢出中断 TOF 产生的次数以及定时器计数器寄存器 TCNTR 的内容进行记录的方法来确定充电时间。当电容 C 充电电压到达 PA0 端口的输入高电平门槛电压 V_{IH}时，计时结束。这时有充电时间 T_s。

在 68HC05K0 中，充电计时可以如下执行：

首先记录在开始充电时 TCNTR 的内容，然后对 TCNTR 产生的溢出中断次数进行记录，最后在结束充电时，再对 TCNTR 的内容进行记录。这样，溢出中断次数等于累计了 TCNTR 满程计数的值；而开始和结束充电时 TCNTR 的值又可以计算出不足满程计数时 TCNTR 的计数值。

例如，TCNTR 产生了 n 次溢出，而在开始计时时 TCNTR 的值为 N_b，在结束计时时为 N_e。

如果有 $N_e>N_b$，则计时时间 T_s 为：

$$T_s=[n\times 1\,024+(N_e-N_b)]\times 4\times T_{OP} \tag{6-8}$$

其中，T_{OP}为内部时钟周期，$T_{OP}=2T_{OSC}$；

T_{OSC}是振荡器周期，$T_{OSC}=1/f_{OSC}$。

如果有 $N_e<N_b$，则计时时间 T_s 为：

$$T_s=[(n-1)\times 1\,024+(1\,024-N_b)+N_e]\times 4\times T_{OP} \tag{6-9}$$

在定标过程中求出的充电时间 T_s 作为定标的时间，用它作为一个参考标准来求取被检测的电阻 R_m 的值。

2. 放电过程

在定标充电过程结束后，马上进入放电过程。

放电时，PB0，PB1 为输入端口，PA0 为输出端口并输出低电平 $V_{OL}=0.3$ V。这时，电容 C 所充的电荷通过 R_d 到 PA0 端进行放电，放电结束有 $V_C=V_{OL}$。

为了加快放电时间，可以把 PB0，PB1 和 PA0 同时置为输出端，并同时输出低电平，则电容 C 通过三个端口同时放电。

3. 检测过程

在检测过程开始时，令 PB0，PA0 为输入端口，PB1 为输出端口并输出高电平。PB1 过 R_m 对 C 的充电时间和定标过程一样，可以用定时器计数器寄存器 TCNTR 的溢出次数及其开始充电计时时的值 N_b 及充电计时结束时的值 N_e 来求得。当电容 C 中的充电电压达到 PA0 端口的输入高电平门槛电压 V_{IH}时，检测过程结束。检测过程的充电时间用 T_m 表示。

在求出了定标充电时间 T_S 及检测充电时间 T_m 之后，就可以求出被测电阻 R_m 的值。

由于在 RC 电路中，充电过程是遵循指数曲线规律的，而且电容 C 充电电压 V_C 和电源电压 E 及 RC 的关系如下：

$$V_C=E(1-e^{-t/RC}) \tag{6-10}$$

从而有

$$1-e^{-t/RC}=\frac{V_C}{E}$$

即

$$e^{-t/RC}=\frac{E-V_C}{E}$$

$$t=R\cdot C\cdot \ln[E/(E-V_C)] \tag{6-11}$$

在上面的定标过程和检测过程中，电容 C 的电压 V_C 在充完电时为 V_{IH}，在放完电时为 V_{OL}。

式(6-11)中，E 相当于 PB0 或 PB1 所输出的高电平，V_C 是电容的电压。故而有

$$V_{IH}=E(1-e^{-t_h/RC}) \tag{6-12}$$

其中，t_h 是电容 C 从 0V 充电到 V_{IH}所用的时间。

显然

$$t_h=R\cdot C\cdot \ln[E/(E-V_{IH})] \tag{6-13}$$

另外，当放电结束时，电容的电压 V_C 等于 PA0 端口输出低电平的门槛电压 V_{OL}。这说明，在充电时电容不是从 0V 开始充电，故而要考虑电容 C 从 0V 充电到 V_{OL}时所需的时间。于是有

$$V_{OL} = E(1 - e^{-t_1/RC}) \tag{6-14}$$

其中，t_1 是电容 C 从 0V 充电到 V_{OL} 所用的时间。

显然有

$$t_1 = R \cdot C \cdot \ln[E/(E - V_{OL})] \tag{6-15}$$

结合式(6-14)和式(6-15)，同时在定标过程中电容 C 是过 R_s 进行充电的，所以有定标充电时间 T_s：

$$\begin{aligned} T_s &= t_h - t_1 \\ &= R_s \cdot C\ln[E/(E - V_{IH})] - R_s \cdot C\ln[E/(E - V_{OL})] \\ &= R_s \cdot C\{\ln[E/(E - V_{IH})] - \ln[E/(E - V_{OL})]\} \\ &= R_s \cdot C \cdot \ln\left[\frac{E}{E - V_{IH}} \Big/ \frac{E}{E - V_{OL}}\right] \\ &= R_s \cdot C \cdot \ln[(E - V_{OL})/(E - V_{IH})] \end{aligned} \tag{6-16}$$

同理，检测充电时间 T_m 为

$$T_m = R_m \cdot C \cdot \ln[(E - V_{OL})/(E - V_{IH})] \tag{6-17}$$

根据式(6-16)和式(6-17)，有

$$\frac{T_s}{T_m} = \frac{R_s \cdot C}{R_m \cdot C} \tag{6-18}$$

从而

$$R_m = \frac{T_m}{T_s} R_s \tag{6-19}$$

很明显，采用式(6-19)很容易求出被检测的电阻 R_m，从而也就容易检测出对应的温度。

通过对充电时间 T_s 和 T_m 的求取，又由于标准电阻 R_s 是已知的，从而可求出 R_m。这个过程也实现了把模拟量转换成数字量的过程，即 A/D 过程。

充电过程的曲线如图 6-18 所示。其中，曲线 1 是定标曲线，用于确定定标时间 T_s；曲线 2 是检测曲线，用于确定检测时间 T_m。有了这两个时间就可以轻而易举地求出 R_m。

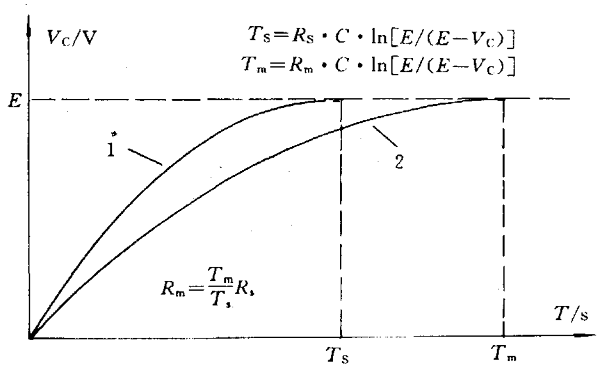


图 6-18　定标和检测过程充电曲线

4. *RC* 电路参数的选择

图 6-17 所示的电路结构中，关键的是要选择恰当的标准电阻 R_s 和电容 C。R_s 和 C 的值选择取决于 A/D 转换所需要的分辨率的位数。

R_s 的取值一般应选取为被检测电阻的最大值的二分之一。设被检测电阻的最大值为

R_{mm}，则 R_s 为

$$R_s = R_{mm}/2 \tag{6-20}$$

电容 C 的取值按下式选取：

$$C = \frac{T_{mm}}{R_{mm} \cdot \ln[(E - V_{OL})/(E - V_{IH})]} \tag{6-21}$$

其中，T_{mm}是检测过程中的最长充电时间；

R_{mm}是被检测电阻 R_m的最大值；

E 是 PB0，PB1 端口输出的高电平；

V_{IH}是 PA0 端口输入高电平的门槛电压；

V_{OL}是 PA0 端口输出低电平门槛电压。

假设 A/D 转换的分辨率是 8 位；68HC05K0 单片机的振荡器振荡频率 $f_{OSC}=2$ MHz，则内部时钟频率 f_{OP}为 1 MHz，内部时钟周期 $T_{OP}=1$ μs。在采用定时器计数器寄存器 TCNTR 的溢出信号中断 TOF 及 TCNTR 的内容作为计时记录，并取 TCNTR 的最多溢出次数为 4 时，T_{mm}的值为

$$T_{mm} = 4 \times 2^8 \times 2^2 \times 1\ \mu s = 4\ 096\ \mu s = 4.096\ ms$$

上式中，2^8 是因为 TCNTR 有 8 位；2^2 是因为在定时器结构中，内部时钟频率 f_{OP}进入 TCNTR 之前经过 2 分频；而 4 是 TCNTR 溢出次数；1 μs 是内部时钟周期 $T_{OP}=1/f_{OP}$。

由于 A/D 转换的分辨率是 8 位，显然，在内部时钟计数为 $T_{mm}/256=16$ μs 时产生一个 A/D转换最低有效位的值，即每 16 μs 为 A/D 转换的分辨率。

考虑被检测的电阻 R_m的最大值为 $R_{mm}=100$ kΩ，PB0，PB1 端口输出的高电平 $E=5$ V，PA0 端口输入高电平的门槛电压 $V_{IH}=3$ V，PA0 端输出低电平时的门槛电压 $V_{OL}=0.3$ V，则电容 C 选取如下：

$$\begin{aligned} C &= \frac{T_{mm}}{R_{mm} \cdot \ln[(E - V_{OL})/(E - V_{IH})]} \\ &= \frac{4.096\ ms}{100\ k\Omega \cdot \ln[(5\ V - 0.3\ V)/(5\ V - 3\ V)]} \\ &= \frac{4.096\ ms}{100\ k\Omega \cdot 0.854\ 4} \\ &= \frac{4.096 \times 10^{-3}\ s}{8.544 \times 10^4 \Omega} \\ &= 0.047\ 940\ 1\ \mu F \end{aligned}$$

在实际中，电容 C 的取值应比计算出来的值要小一些，以保证在测量时单片机不会产生计数值超越所设计的最大值的情况。所以，电容 C 的值可取 0.047 μF。

这样，在图 6-17 所示的 A/D 转换结构中，就有如下参数：$R_s=50$ kΩ，$R_d=620$ Ω，$C=0.047$ μF，R_m 的最大电阻值为 100 kΩ。

在实际中，随着温度的变化，电阻的阻值和电容容量都会产生一些偏移；而电容的容量值和标称值之间也有一定的误差。在定标过程中，可以求出定标时间 T_s；以 T_s 为参考标准，就可以消除所有的一阶误差，包括偏移、电容不准、电源误差等。

一般应选择低漂移的电阻，同时，把电阻的阻值放在内存中以减少测量误差；在软件中可

以采用一些补偿方法。其他一些产生误差的来源是 I/O 端口的漏电流，电阻和电容的非线性，输入门槛电压的不确定和测量时间的不确定。测量时会有正负起码一个内部时钟周期的时间误差。

5. 控制软件框图

无偏置电源的 *RC* 实现 A/D 转换方法需要软件进行控制。控制软件框图如图 6-19 所示。

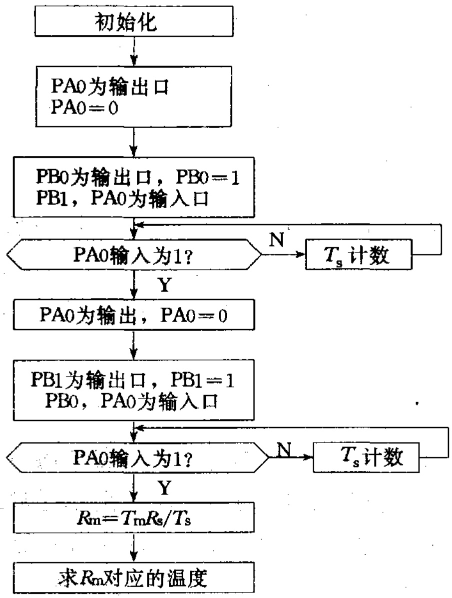


图6-19 无偏置电源 *RC* 实现A/D的控制软件框图

图 6-19 所示的控制软件框图是依据图 6-17 所示的结构通过 *RC* 电路实现 A/D 转换方法的程序流程。

图 6-17 中，采用了 3 个 I/O 端口。其中，PA0 专门用于放电及检测电容 *C* 的充电电压。这样做是为了无论 R_m 还是 R_s 过 *C* 充电，都能具有相同的高电平门槛电压对 V_C 进行检测，也以同样低电平门槛电压使 *C* 放电结束，以保证 R_m,R_s 充放电的门槛一致性，从而保证检测的精确性。

从简单的角度讲，采用 PB0,PB1 端口也足够了，因为在 R_s 过 *C* 充放电时，可以用 PB1 的门槛电压进行检测。在 R_m 过 *C* 充放电时，可以用 PB0 的门槛电压进行检测。但由于 PB0 和 PB1 的门槛电压是不一样的，这就会给检测带来误差。故而专门采用一个 I/O 端口 PA0 对 R_m,R_s 过 *C* 充放电进行检测。

开始时，程序对 68HC05K0 进行初始化。此时分配内存，安排有关数据，定义中间寄存单元等。接着，对电容 *C* 进行放电。故令 PA0 为输出口，并输出低电平，此时电容 *C* 可过 R_d 和 PA0 放电，放电的结果令电容的电压 $V_C=V_{OL}$。随后进入定标充电过程。这时，令 PB0 为输出口，PB1,PA0 为输入口，同时 PB0 输出高电平，这个高电平过 R_s 对 *C* 充电。

程序检测 PA0 的输入电压，如果不为 1(即高电平)，则 TCNTR 进行计数。在进入定标充电过程的瞬间，程序把 TCNTR 的内容送入内存的特定单元存放。TCNTR 是在其基础上不断进行计数的。当 PA0=1 时，把 TCNTR 的当前值取出，并和 TCNTR 原有内容、溢出次数合起来处理，求出 T_s。

接着进入放电过程，把 PA0 设置为输出口，并令 PA0=0，此时电容 C 的电压 V_C 过 R_d 放电，又有 $V_C=V_{OL}$。

放电结束后进入检测充电过程。这时，程序令 PB1 为输出口，PB0，PA0 为输入口，并且 PB1 输出高电平过 R_m 对 C 进行充电。

充电开始瞬间，程序把 TCNTR 的内容存入内存特定单元，并对 PA0 的输入电平进行检测。如果 V_C 充电不是逻辑"1"，则 TCNTR 继续计数；如果 $V_C=1$，则把此时 TCNTR 的内容取出来，和先前存入内存的 TCNTR 的内容、TCNTR 的溢出次数一起加以处理，求出 T_m。

由于已求出了 T_s 和 T_m，所以利用已知的标准电阻值 R_s，通过程序计算出 R_m。计算时采用公式 $R_m=T_mR_s/T_s$。

在求出 R_m 之后，可以根据热敏电阻的分度表查出对应的温度。至此，就实现了温度的 A/D转换。

二、有偏置电源的 RC 实现 A/D 方法

有偏置电源的 RC 实现 A/D 转换，就是在单片机外部的 RC 电路中加有专门的偏置电源，通过 RC 充放电过程，在单片机程序控制下实现 A/D 转换的方法。由于外部有专门的偏置电源，所以，实现 A/D 转换就可以只用 2 个 I/O 端口，而不是像无偏置电源的 RC 实现 A/D 的方法那样需要 3 个 I/O 端口。

有偏置电源的 RC 实现 A/D 方法的电路结构如图 6-20 所示，其中 R_x 是检测的热敏电阻。

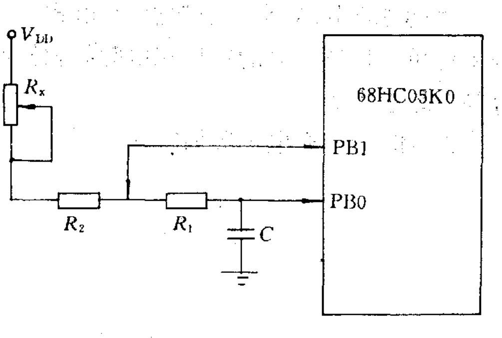


图 6-20 有偏置电源的 RC 实现 A/D 电路

在图 6-20 中，只用到 68HC05K0 的 PB1 和 PB0 端口。电容 C 直接接在 PB0 和地之间；电阻 R_1 接在 PB1 和 PB0 之间，R_2 和 R_x 串联在 PB1 和电源 V_{DD} 之间。电源 V_{DD} 就是 RC 的偏置电源。

下面说明这个有偏置电源 V_{DD} 的 RC 实现 A/D 的方法。在整个过程中，PB0 端口为输入口。

1. 定标过程

在定标开始时，首先要把电容 C 中的电荷释放掉。这时，PB1 输出低电平 V_{OL}，则电容 C 过 R_1 放电。最后令电容 C 中的电压 $V_C=V_{OL}$。

接着，PB1 端口输出高电平并达 V_{DD}，过 R_1 对 C 充电。当电容 C 充电电压到达 PB0 的输入高电平门槛电压 V_{IH} 时，则停止充电。单片机 68HC05K0 记录这段充电时间 T_s。

T_s 实质是定标充电时间，从前面式(6-13)和式(6-14)可知

$$T_s = R_1 \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-22}$$

2. 放电过程

在检测出 T_s 之后，则 PB1 又输出低电平 V_{OL}，电容 C 又过 R_1 对 PB1 放电，最后有 $V_C=V_{OL}$。

3. 检测过程

这时，PB1 变为输入端口，故而输出是高阻抗。则偏置电源 V_{DD} 过 R_x，R_2，R_1 对电容 C 充

电。当充电使电容 C 的电压 V_C 到达 PB0 输入高电平门槛电压 V_{IH} 时，则充电结束。单片机记录这段充电时间 T_m。

T_m 实质上是检测充电时间，表示如下：

$$T_m = (R_x + R_2 + R_1) \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-23}$$

为了简单起见，设有 $R_2=R_1=R$，则有

$$T_m = (R_x + 2R) \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-24}$$

令
$$T_f = T_m - 2T_s \tag{6-25}$$

则有

$$T_f = R_x \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-26}$$

在式(6-26)中，由于电容 C 可以固定，而 V_{DD},V_{IH},V_{OL} 都是可知的，所以，为了简化，令

$$K = C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-27}$$

则式(6-26)变为

$$T_f = R_x K \tag{6-28}$$

最后有

$$R_x = T_f/K \tag{6-29}$$

式(6-29)中，K 是一个常数，它由式(6-27)求出；而 $T_f=T_m-2T_s$，T_m 和 T_s 都是在 A/D 过程中由单片机测出来的定标充电时间和检测充电时间，故而 T_f 很容易求出来。这样，就可以十分简单地求出 R_x。

T_m 和 T_s 的充电波形如图 6-21 所示。

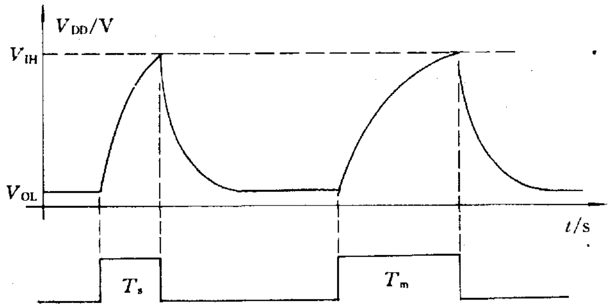


图 6-21 T_m,T_s 充电波形及时间

4. *RC* 电路的参数选择

如果把定时器计数器寄存器 TCNTR 的溢出作为最大的时间界限，那么，在 2 MHz 的振荡频率中，68HC05K0 的内部时钟频率 f_{OP} 为 1 MHz，故它的周期为 1 μs。由于内部时钟频率 f_{OP} 要经过 4 分频后才对 TCNTR 计数，而 TCNTR 有 8 位，所以，在 TCNTR 产生溢出时一共计数了 2^{10}=1 024个内部时钟，也即在 TCNTR 产生溢出时，其计数时间为 1.024 ms。

只要在 TCNTR 产生溢出时开始进行 A/D 转换，那么，这时 TCNTR 会为全“0”状态。从该时刻开始计时到下一次 TCNTR 产生溢出，就会有 1.024 ms 时间。

为了在本次 TCNTR 溢出时开始进行 A/D 转换，而在下次 TCNTR 溢出之前结束 A/D 转换，最大计时时间应限制在 1.0 ms 之内。

这样，实质上要求 68HC05K0 外部的 RC 回路最大充电时间方程为

$$(R_x + R_1 + R_2) \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] < 1 \text{ ms} \tag{6-30}$$

为简化运算过程，设有最大时间 T_{max}，最大输入高电平门槛电压 V_{IHmax}，最大回路电阻值 R_{max}，最大输出电平门槛电压 V_{OLmax}，则有

$$R_x <= R_1 = R_2 = R_{max} = R$$

$$V_{IHmax} = 0.7 \cdot V_{DD}$$

$$T_{max} = 1 \text{ ms}$$

$$V_{OLmax} = 0.3 \text{ V}$$

把这些值代入式(6-30)中，则有

$$\begin{aligned} &3RC \cdot \ln[(V_{DD} - 0.3)/(V_{DD} - 0.7V_{DD})] \\ &= 3RC \cdot \ln[4.7/1.5] \\ &= 3RC \cdot \ln[3.1333] \\ &= 3RC \cdot 1.141 \\ &= 3.423RC < 1 \text{ ms} \end{aligned} \tag{6-31}$$

电阻 R 受到两种因素的限定：

(1) I/O 的输出阻抗应大大小于负载电阻 R；

(2) R 不能太大，太大则来自输入端口的漏电流会影响结果的正确性。

根据这两个原则，假设 I/O 端口在输出高电平时提供 1 mA 充电电流，则这时 I/O 端口的阻抗为 5 V/1 mA=5 kΩ。

第一个限制要求外部电阻远大于 I/O 端口阻抗，这时可取 R=20 kΩ。

另外，当 I/O 端口为输入端口时，其最大漏电流小于 10 μA。这时，输入 I/O 端口的阻抗为 5 V/10 μA=500 kΩ。

第二个限制要求 I/O 端口输入状态的漏电流不能影响结果，则要求 R≪500 kΩ。显然，取 R=20 kΩ 满足此条件。

由式(6-31)得

$$3.423 \cdot 20 \text{ k}\Omega \cdot C < 1 \text{ ms}$$

即

$$C < \frac{1}{68\ 460} \text{ ms}/\Omega$$

$$C < 0.0146\ \mu\text{F} \tag{6-32}$$

为了保证计数时产生的计时时间小于 1 ms，C 取 0.01 μF。于是得到最后结果为

$$R_1 = R_2 = R = 20 \text{ k}\Omega$$

$$R_x <= 20 \text{ k}\Omega$$

$$C = 0.01\ \mu\text{F}$$

5. 运算结果分析

图 6-22 中给出了这种 A/D 转换方法所产生的结果。

从图 6-22 中可以看出，检测特性有十分良好的线性，同时也存在 5%的偏差。偏差并不是由外部元件的值(R,C 的值)引起的，产生这种偏差的原因在于 PB1 端口在输出高电平时一般不可能高于 4.9 V。在定标过程中，PB1 输出的高电平不高于 4.9 V 就会产生这种偏差。

在定标过程中，PB1 输出的高电平低于 V_{DD}=5 V，则可以认为在 68HC05K0 的 PB1 端口

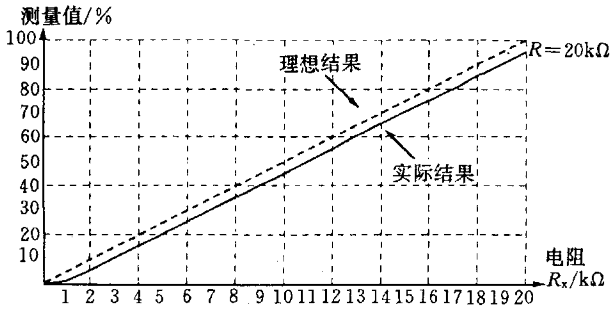


图 6-22 电阻 R_x 的检测结果

串了一个电阻 R_h。该电阻是 PB1 输出高电平时的内部串联电阻。当进行定标充电时，由于对 C 充电电流也流过 R_h，故而产生压降，使 PB1 输出的电平下降，所以，这时输出的高电平达不到 $V_{DD}=5$ V。

很明显，要消除这种 A/D 转换偏差，就应想办法消除 R_h 的影响。因此，就要采用特别的方法进行 A/D 转换。

6. 改进 A/D 转换的第一种方法

考虑在定标过程中 PB1 端口内部存在串联电阻 R_h，此时的定标时间 T_s 为

$$T_s=(R_1+R_h)\cdot C\cdot\ln[(V_{DD}-V_{OL})/(V_{DD}-V_{IH})] \quad (6\text{-}33)$$

为了消除 R_h 的影响，在外部电路中加上一个电阻 R_n，这时的电路结构如图 6-23 所示。

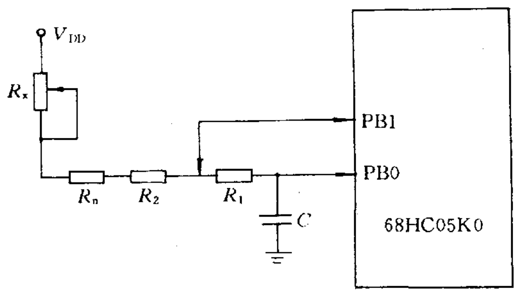


图 6-23 改进 A/D 的第一种结构

在加上了 R_n 之后，检测过程的检测充电时间 T_m 为

$$T_m=(R_x+R_n+R_2+R_1)\cdot C\cdot\ln[(V_{DD}-V_{OL})/(V_{DD}-V_{IH})] \quad (6\text{-}34)$$

由于

$$T_f=T_m-2T_s$$

$$R_1=R_2=R$$

$$K=C\cdot\ln[(V_{DD}-V_{OL})/(V_{DD}-V_{IH})]$$

则有

$$T_f=[(R_x+R_n+2R)-2(R+R_h)]\cdot K$$
$$=(R_x+R_n-2R_h)\cdot K \quad (6\text{-}35)$$

为了消除 R_h 的影响，并保证 T_f 和 R_x 有式(6-28)所示的关系，即

$$T_f=R_x\cdot K$$

故而，对 R_n 取值为

$$R_n=2R_h \quad (6\text{-}36)$$

实际上，PB1 端口内部的串联电阻 R_h 是通过实际检测求出的。为了测量出 R_h，用程序控制使 PB1 输出高电平，并且在外部接一个电阻到地，根据外接电阻的大小以及 PB1 端口输出的电平值，就可以知道其内部串联电阻 R_h 的值。例如，外接一个 4.7 kΩ 电阻到地，而 PB1 端

口输出的高电平是 4.7 V，则说明 $R_h = 0.3\ k\Omega$。一般可以测出，PB1 端口输出高电平的内部电阻 $R_h = 0.2\ k\Omega \sim 0.35\ k\Omega$。所以，在外部 A/D 的 RC 电路中，新增加的电阻 $R_n = 0.4\ k\Omega \sim 0.7\ k\Omega$。

在加了 R_n 之后，A/D 的特性和测量的结果如图 6-24 所示。在该特性中，$R_n = 680\ \Omega$。

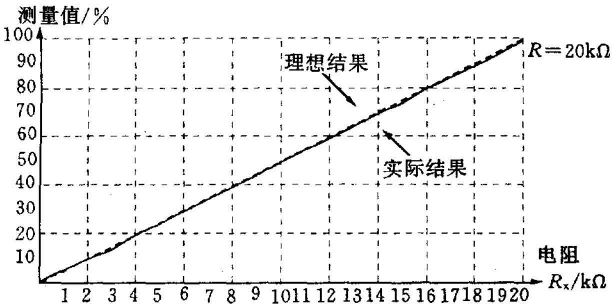


图 6-24 外部 RC 电路加了 R_n 的改进结果

7. 改进 A/D 转换的第二种方法

为了改进由于 PB1 端口内部存在的串联电阻 R_h 的影响，可以考虑用另一个端口输出的高电平取代电源 V_{DD}。由于另一个端口输出高电平时，其内部也存在串联电阻，而在单片机内部，端口的内部串联电阻是相等或很接近的，显然，用另一个端口输出高电平取代 V_{DD} 对 R_x 回路充电就可以消除误差。

这种改进方法的电路结构如图 6-25 所示。图中用 PA0 端口输出的高电平取代 V_{DD}。

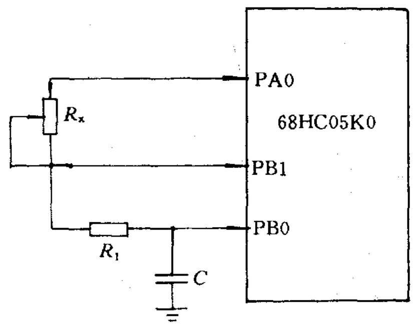


图 6-25 改进 A/D 的第二种结构

图 6-25 中也取消了 R_2，这主要是为了求 T_f 方便，另外也简化了电路。

这时，定标过程的充电时间 T_s 为

$$T_s = (R_1 + R_h) \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-37}$$

PA0 端口的内部串联电阻用 R_n 表示，则测量过程的充电时间 T_m 为

$$T_m = (R_x + R_n + R_1) \cdot C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})] \tag{6-38}$$

在这种方法中，令

$$T_f = T_m - T_s \tag{6-39}$$

则有

$$T_f = (R_x + R_n - R_h) \cdot K \tag{6-40}$$

其中，

$$K = C \cdot \ln[(V_{DD} - V_{OL})/(V_{DD} - V_{IH})]$$

只要

$$R_n = R_h \tag{6-41}$$

则有

$$T_f = R_x \cdot K \tag{6-42}$$

于是便可消除 R_h 的影响。

在实际中，同一个单片机中的不同端口在输出高电平时，其内部串联电阻也不一定相等，而只是接近，故而要彻底清除 R_h 是不可能的。但是，这种方法也取得了良好的效果。图 6-26 是这种方法所得 R_x 检测的结果。从图中看出，它的实际结果与理想结果基本吻合。

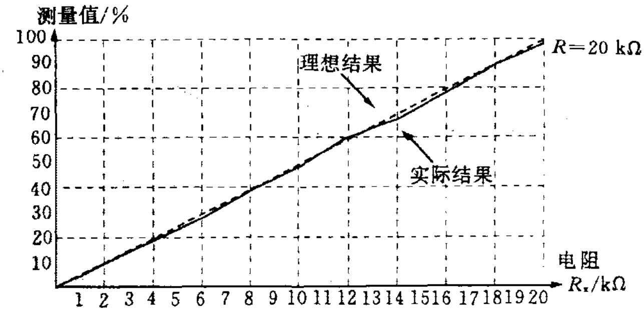


图 6-26　改进方法二所得的结果

三、含运算放大器的 *RC* 实现 A/D 方法

前面讲的无偏置电源和有偏置电源的 *RC* 实现 A/D 方法并不是一种通用的 A/D 转换方法，它们只是在被测电阻 R_x 是热敏电阻或其他可变电阻时，可以随时检测出 R_x 的现行电阻值。

要构成一个通用的 A/D 转换器，把输入的电压大小转换成相对应的数字量，就需要采用含运算放大器的 *RC* 电路。

含运算放大器的 *RC* 电路实现 A/D 的结构如图 6-27 所示。它在单片机 68HC05K0 外部由一个运算放大器 U_1，一个电阻 R 和一个电容 C 组成。

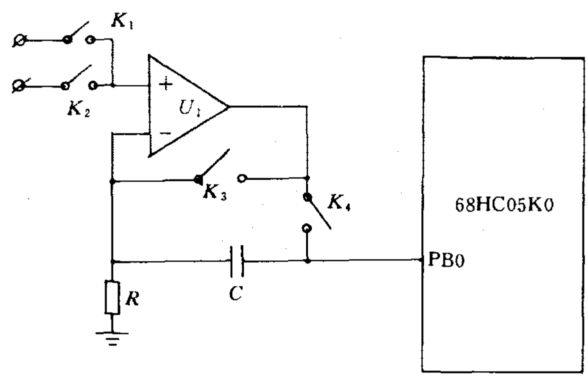


图 6-27　含运放及 *RC* 的 A/D 结构

这种 A/D 结构由硬件和软件结合产生分辨率从 6 位到 10 位的二进制数，转换时间一般为数百微秒；被转换的模拟量可以是电流或电压。在软件中可以采用软件校正技术以补偿时间和温度漂移所产生的误差。

1. A/D 工作原理

在图 6-27 中，A/D 转换器实质上由 5 个器件组成，分别是单片机 68HC05K0、运算放大器

U_1、电容 C、电阻 R 以及开关 $K_1 \sim K_4$。其中，$K_1 \sim K_4$ 是一块模拟开关集成电路中的开关电路，是由 68HC05K0 的 PA 端口控制的。

在 68HC05K0 的软件控制之后，开关 $K_1 \sim K_4$ 以不同的组合形式控制电容 C 放电及充电过程，从而完成 A/D 转换。A/D 转换分为两个阶段，即放电过程和输入电压测量过程。在运算放大器 U_1 的输入端有开关 K_1 和 K_2，K_2 接参考电压，K_1 接被测电压。实际上，A/D 转换过程分 4 个过程进行，即放电、测量参考电压、放电和测量被测电压。

下面分别介绍这 4 个过程：

第一个放电过程

放电过程的目的是使电容 C 中的电荷对地释放掉，以准备进行外部电压测量。第一个放电过程是为测量参考电压做准备。

在放电过程中，K_3 接通，K_4 关断，同时，PB0 输出低电平，于是，电容 C 通过 PB0 到地以及电阻 R 进行放电。这时，电路的等效电路如图 6-28 所示。

测量参考电压

在测量参考电压时，K_1，K_3 关断；K_2，K_4 接通。参考电压通过 K_2 连接到运算放大器 U_1 的正输入端，同时，K_4 接通使 U_1 的输出端和负输入端连接在一起。这时，电路的等效电路如图 6-29所示。电路输出的电流是输入电压的线性函数。

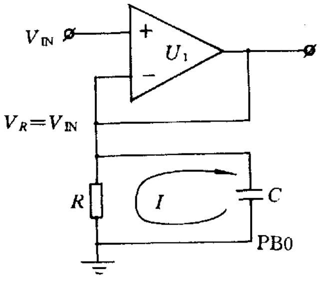


图 6-28　放电等效电路

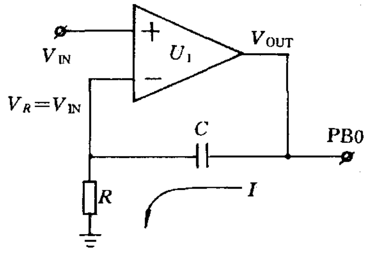


图 6-29　测量电压时的充电等效电路

这时，输入电压 V_{IN}加到 U_1 的正输入端，则 U_1 的输出对 RC 回路的电容 C 充电。同时，PB0 变成输入端口，而 68HC05K0 的内部定时器开始进行计数定时。

当电容 C 充电达到 PB0 输入高电平的门槛电压时，定时器停止计数。这时，定时器计数的结果反映了输入电压的大小。由于输入电压是参考电压 V_{ref}，故而这时是测量参考电压的过程。

第二个放电过程

这时，K_3 接通，K_4 关断，PB0 变成输出端口并输出低电平。电容 C 过 PB0 和电阻 R 放电，并为测量被测电压做准备。放电等效电路如图 6-28 所示。

测量被测电压

这时，K_2，K_3 关断；K_1，K_4 接通。被测电压 V_m 通过 K_1 连接到运算放大器 U_1 的正输入端。由于 K_4 接通，电路的等效电路如图 6-29 所示。

在 K_4，K_1 接通瞬间，电容 C 就会充电。这时 PB0 变为输入端口，同时，定时器进行计数。

当电容 C 充电达到 PB0 的输入高电平门槛电压时，定时器停止计数。这时，定时器计数结果代表了被测电压 V_m 的大小。这个过程就是测量被测电压的过程。

2. 电路元件的选择

从图 6-29 中可以看出，对输入电压 V_{IN}进行测量之前，因电容 C 放了电，故有 $V_C=0V$。这就相当于运算放大器 U_1 的输出端与负输入端相连短路。这时，运算放大器 U_1 相当于一个跟随器，其输出电压等于正输入端电压，即 $V_{OUT}=V_{IN}$。于是输出电压过 R 对 C 充电，充电电流 $I=V_{OUT}/R$，即 $I=V_{IN}/R$。

根据运算放大器的性质，其负输入端电位和正输入端电位近似相等，因此有 $V_R \approx V_{IN}$。

这样，在电容 C 充电过程中便有以下特点：

(1) 充电过程中电流不变，即 $I=V_{IN}/R$；

(2) 运算放大器 U_1 的输出电压 V_{OUT}等于负输入端电压加上电容 C 电压，即

$$V_{OUT} = V_R + V_C = V_{IN} + V_C \tag{6-43}$$

(3) 电容 C 充电电流 I_C 和流过电阻的电流 I_R 相等。

下面分析 A/D 转换过程中，输出电压 V_{OUT}和时间 t 的关系。

根据电容的特性，流过电容的电流可以用下式表示：

$$I_C = C\frac{dV_C}{dt} \tag{6-44}$$

流过电阻的电流 I_R 用下式表示：

$$I_R = V_R/R \tag{6-45}$$

在图 6-29 中显然有

$$V_R = V_{IN} \tag{6-46}$$

$$I_R = I_C = I \tag{6-47}$$

从而有

$$\frac{V_{IN}}{R} = C\frac{dV_C}{dt} \tag{6-48}$$

$$V_C = \int \frac{V_{IN}}{RC}dt = \frac{V_{IN}}{RC}t \tag{6-49}$$

同时，由于有

$$V_{OUT} = V_C + V_R$$

即

$$V_{OUT} = V_C + V_{IN} \tag{6-50}$$

则有

$$\frac{V_{IN}}{RC}t + V_{IN} = V_{OUT} \tag{6-51}$$

$$V_{IN}\left(\frac{t}{RC} + 1\right) = V_{OUT}$$

$$t = \left(\frac{V_{OUT}}{V_{IN}} - 1\right) \cdot RC \tag{6-52}$$

也可写成

$$t = (V_{OUT} - V_{IN}) \cdot RC/V_{IN} \tag{6-53}$$

对于输入电压V_{IN}的 A/D 转换过程，考虑输出电压V_{OUT}到达 PB0 输入高电平门槛电压V_{IH}时则转换过程结束，即有转换时间t_{AD}

$$t_{AD} = (V_{IH} - V_{IN}) \cdot RC/V_{IN} \tag{6-54}$$

其中，V_{IH}是 PB0 输入高电平门槛电压；

V_{IN}是被转换的输入电压；

R 是充电回路电阻；

C 是充电回路电容。

从式(6-54)看出，RC/V_{IN}是常数，V_{IH}是常数，则被测电压V_{IN}越大，t_{AD}越小；V_{IN}越小，则t_{AD}越大。特别有$V_{IN}=V_{IH}$时，$t_{AD}=0$；而$V_{IN}=0$时，$t_{AD}=V_{IH}\cdot RC/V_{IN}$，为最长转换时间。

实际上，V_{IN}是不会等于 0 的，如果V_{IN}等于 0，则转换时间 t 为无穷大。

设当V_{IN}很小且相对于V_{IH}的值可忽略不计时，则式(6-54)可写为

$$t = V_{IH} \cdot RC/V_{IN} \tag{6-55}$$

从而有

$$R \cdot C = t \cdot V_{IN}/V_{IH} \tag{6-56}$$

其中，V_{IN}是被测量的电压；

V_{IH}是 PB0 输入高电平时的门槛电压；

t 是 A/D 转换时间。

根据式(6-56)，可以对 RC 的值进行选择。

设V_{IN}的最小值为 100 mV，PB0 输入高电平门槛电压$V_{IH}=3$ V。采用定时器计数器寄存器 TCNTR 进行计数。当振荡频率为 2 MHz 时，它产生溢出的计数时间为 1 024 μs。为了防止溢出，取 t=1 ms。根据式(6-56)有

$$\begin{aligned} R \cdot C &= 1\ \text{ms} \cdot 100\ \text{mV}/3\ \text{V} \\ &= 1\ \text{ms} \cdot 0.033 \\ &= 33\ \mu\text{s} \end{aligned}$$

取 $R = 33\ \text{k}\Omega$

则有 $C = 33\ \mu\text{s}/33\ \text{k}\Omega = 1\ 000\ \text{pF}$

由于 PB0 输入高电平门槛电压V_{IH}为 3 V，则输入的被测量电压V_{IN}是不能高于 3 V 的。因此，对于输入高于 3 V 的被测电压，需要在运算放大器的正输入端加上电阻分压网络。分压的结果只要能使运算放大器 U_1 正输入端的输入电压小于 3 V 即可。

至于参考电压，应该选取在 2 V 左右，这样有利于提高 A/D 转换的性能。

参考电压测量的过程也称校正周期。用于消除所有的一阶误差，包括偏移、增益、R 和 C 的偏差、电源电压和温度偏差。当然，参考电压偏差是不能消除的。

参考电压用V_{ref}表示，对其测量所得的时间为t_{ref}；被测电压用 V_m 表示，对其所测得的时间为 t_m，则有

$$\frac{V_m}{V_{ref}} = \frac{t_m}{t_{ref}} \tag{6-57}$$

从式(6-57)可知，如果在测量 V_m 时存在某些偏差，则这些偏差在测V_{ref}时也会存在，并且分别反映在 t_m 和 t_{ref}上。根据式(6-57)有

$$V_m = \frac{t_m}{t_{ref}} \cdot V_{ref} \tag{6-58}$$

利用 t_m，t_{ref}和 V_{ref}就可以求出 V_m。而 t_m 和 t_{ref}中的偏差类同，故在式(6-58)中它们会对消有差偏差，从而提高了精确度。

图 6-27 中，开关 K_1，K_2，K_3 和 K_4 是模拟开关，在实际应用中采用模拟开关集成电路。模拟开关的漏电流、电阻和电容的非线性，输入高电平门槛电压的不确定性，都对测量的过程产生影响，从而导致测量误差。在应用中，选择漏电流小的模拟开关也是很重要的。

采用运算放大器所构成的 A/D 转换器一般有较好的转换精度，精度一般可达全标度的 1%。

实际上，这种 A/D 转换器可以组成位数更多的转换器。按用户需要可设计成 10 位、12 位等有较高位数的转换结果。

6.3.3 温度检测仪电路及工作原理

温度检测仪的电路原理图如图 6-30 所示。它由电源稳压电路、采样及 A/D 电路、振荡电路、显示电路和单片机 MC68HC05K0 等组成。

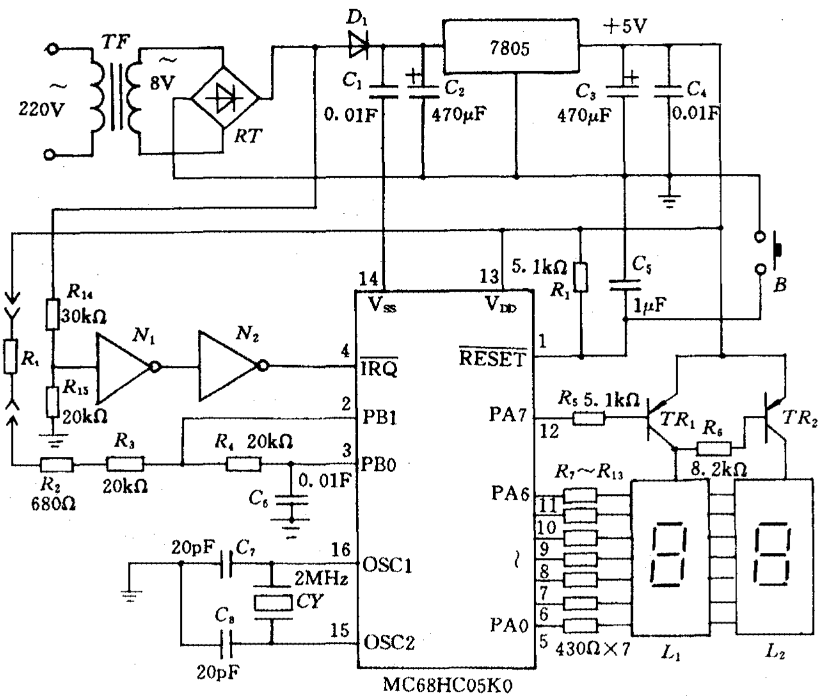


图 6-30 温度检测仪电路原理图

电源稳压电路由变压器 TF、整流器 RT、隔离二极管 D_1、稳压集成电路 7805 及滤波电容等构成。变压器 TF 把工频 220 V 交流电压降压成 8 V 的低压，以便随后实现低压稳压。8 V 低压交流电由整流环节 RT 整流成直流电压。这时的直流电压是脉动直流电压。要使脉动直流电压平滑就要进行滤波。二极管 D_1 的作用是为了把脉动直流电压和滤波平滑直流电压隔离开来。整流后的脉动直流电压可以用于产生过零脉冲，从而可用于中断，进而用于计时、显示扫

描或可控硅触发同步。在这里主要用于显示扫描同步。脉动直流电压通过 D_1 之后由 C_1,C_2 进行滤波,从而产生较平滑的直流电压,再由稳压集成电路 7805 调整之后,输出稳定的 5 V 直流电压,为了使 5 V 直流电压有较好的瞬态负载能力,在 7805 的输出端也加上电容 C_3,C_4 滤波环节。

N_1 和 N_2 两个反相器用于形成过零脉冲。脉动直流电压通过两个反相器 N_1,N_2 后被放大和整形,产生和脉动直流电压同步的过零脉冲,并加到单片机的外部中断请求端 $\overline{IRQ}$。过零脉冲形成的图形如图 6-31 所示。

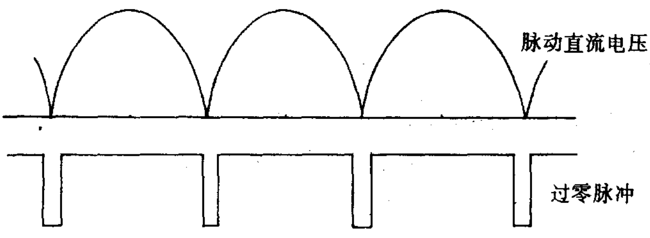


图 6-31　过零脉冲

R_t 是热敏电阻,用于检测温度。R_t,R_2,R_3,R_4 和 C_6 组成了有电源偏置的 RC 实现 A/D 转换的结构。它的电路原理如图 6-23 所示。其中,R_2 相当于图 6-23 中的 R_n,用于克服 PB1 输出高电平时内部串联电阻所引起的偏差。

R_t,R_2,R_3,R_4 和 C_6 组成的 A/D 结构分 4 步执行工作:

(1) PB1 输出低电平,C_6 通过 R_4 对 PB1 进行放电,最终使 C_6 的电压趋于 0;

(2) PB1 输出高电平,通过 R_4 对 C_6 充电,单片机开始对充电计时;当 C_6 的电压达到 PB0 输入高电平门槛电压时,停止对充电的计时,这时充电经历的时间 T_s 称为定标时间;

(3) PB1 输出低电平,C_6 通过 R_4 对 PB1 进行放电,最终令电容 C_6 的电压趋于 0;

(4) PB1 变为输入端,故输出高阻抗,则+5 V 电源通过 R_t,R_2,R_3,R_4 对电容 C_6 充电,单片机开始对充电过程进行计时;当 C_6 的电压达到 PB0 输入高电平门槛电压时,单片机停止计时,这时的充电时间 T_m 称为检测时间。

取 $T_f = T_m - 2T_s$

则有

$$R_t = T_f/K$$

其中,$K = C_6 \cdot \ln[(5-V_{OL})/(5-V_{IH})]$;

V_{OL}是 PB1 输出低电平;

V_{IH}是 PB0 输入高电平门槛电压。

当 $V_{OL} \approx 0$ V,$V_{IH} \approx 3$ V 时,有

$$K = C_6 \cdot \ln[5/2] = C_6 \cdot \ln 2.5 = 0.01\ \mu\text{F} \cdot 0.916\ 3 = 0.916\ 3 \times 10^{-8}\ \text{F} \quad (6\text{-}59)$$

则有

$$R_t = T_f/K = T_f \cdot 1.09 \times 10^8\ \text{F}^{-1} \quad (6\text{-}60)$$

在图 6-30 所示电路采用的 RC 实现 A/D 的结构中,要求 $R_t \leqslant R_3 = R_4$,即 $R_t \leqslant 20$ kΩ。所以,在实际应用时,要采用额定零功率电阻 R_{25}的值小于 20 kΩ 的热敏电阻。

振荡电路由石英晶体 CY,电容 C_7,C_8 组成。石英晶体采用 2 MHz 的频率,故 68HC05K0

的内部时钟频率为 1 MHz。在这种频率下工作时，定时器计数器寄存器 TCNTR 计数产生溢出需要 1 024 μs。实际上，振荡电路也可以用 RC 电路组成，只是由于阻容振荡器容易受温度影响，振荡频率不会稳定于一个频率上，故在这里未采用。实质上对于温度检测而言，一般用阻容 RC 振荡器也可以满足要求。

R_1 和 C_5 组成了上电复位电路，按钮 B 用于手动复位。在接通电源的瞬间，由于+5 V 电源通过 R_1 对 C_5 充电，故而$\overline{\text{RESET}}$端的电位有一个阶段仍处于低电平，从而对 68HC05K0 实现复位。当用户需要复位系统时，用手按动 B，使$\overline{\text{RESET}}$端和地短路，由于人按动按钮要花几十毫秒时间，从而实现了复位。

显示电路由晶体管 TR_1，TR_2，7 段发光二极管显示器 L_1，L_2 以及电阻 R_5～R_{13}组成。其中，R_7～R_{13}是显示器的限流电阻，R_5，R_6 分别是晶体管 TR_1，TR_2 的偏置电阻。TR_1，TR_2 是开关，用于选择 L_1 或 L_2 显示器。显示器是采用扫描方式工作的。每当中断输入端$\overline{\text{IRQ}}$有过零脉冲，则对 L_1，L_2 的显示情况进行一次位转换。假定在第 k 个过零脉冲到来时，L_1 显示，则 L_2 不显示，而在第 $k+1$ 个过零脉冲到来时，L_1 不显示，L_2 进行显示。L_1 和 L_2 的转换是由 PA7 输出的电平信号控制的。PA7=0，选中 L_1，这时 TR_1 晶体管导通，TR_2 截止；PA7=1 时，TR_1 截止，TR_2 导通，选中 L_2 显示。由于过零脉冲周期为 10 ms，所以，L_1 和 L_2 都在 1 s 之内显示 50 次，看起来 L_1 和 L_2 都是连续亮的。

为了保证 L_1，L_2 显示器有一定的亮度，要求流过每一段数码管的电流有一定的大小。因此，对 R_7～R_{13}的每个电阻有如下要求：

$$R \leqslant \frac{V_P - V_{TR} - V_d}{I_d} \tag{6-61}$$

其中，V_P 是电源的直流电压，$V_P=5$ V；

V_{TR}是晶体管导通压降，$V_{TR}=0.3$ V；

V_d 是数码管中一段压降，$V_d=1.2$ V；

I_d 是流过数码管中某段的电流，5 mA$<I_d<$10 mA，既保证亮度又节约能源。

一般取数码管中每段电流 $I_d=8$ mA，则式(6-61)成为

$$R \leqslant (5 - 0.3 - 1.2)\text{V}/8\text{ mA}$$

$$R \leqslant 437.5\ \Omega$$

最后取

$$R = 430\ \Omega$$

温度检测仪的工作原理如下：

当接通电源时，R_1，C_5 组成的复位电路产生复位信号送到 68HC05K0 的$\overline{\text{RESET}}$端，68HC05K0 复位，程序从首部开始执行。首先对单片机进行初始化工作，接着对 R_t 进行 A/D 转换，转换之后得到 R_t 的电阻值，并根据 R_t 的值求出对应的温度。68HC05K0 的内存容量很小，片内 RAM 只有 32 个字节，而且还要用于存放数据和作为堆栈。另外，中断时要将 CPU 中的有关寄存器内容压入堆栈，这些寄存器有条件码寄存器 CCR、累加器 A、变址寄存器 X 和程序计数器 PC 共 4 个，所以在堆栈中会占用 5 个存储单元。如果有两重中断嵌套，就要占用 10 个存储单元。考虑到程序在调用子程序时也要把返回地址，即程序计数器内容压入堆栈，还要占用 2 个存储单元，所以，在考虑一定的余度情况下需要保留 14 个存储单元为堆栈区。这样，存放数据的 RAM 就只剩下 18 个存储单元了。显然，用于存放一张热敏电阻的分度表是不可

能的。所以，只能在用户 ROM 中存放热敏电阻分度表中的主要数据，并依据这些主要数据进行分段线性化处理，按插值方法求得 R_t 值相应的温度值。

在实际中，由于要求 R_t 的最大值小于 20 kΩ，所以，在温度检测仪的测量范围为 10 ℃～99 ℃的情况下，选用额定零功率电阻值为 10 kΩ 的热敏电阻，即 $R_{25}=10$ kΩ。也即是说，在温度为 25 ℃时，其电阻值为 10 kΩ。由于它是具有负温度系数的，故而温度越高，电阻值越低，从而满足了 $R_t<20$ kΩ 的条件。

在选择 R_t 时，所选的热敏电阻可以用 $R_{25}=10$ kΩ，$B_{25/85}=3\ 650$ K 的产品。在表 6-2 中给出了这种热敏电阻在 0 ℃～100 ℃时的分度表。

表 6-2　$R_{25}=10$ kΩ，$B_{25/85}=3\ 650$ K 的热敏电阻 0 ℃～100 ℃分度表

T/℃	R/kΩ	T/℃	R/kΩ	T/℃	R/kΩ
0.0	29.942 0	34.0	6.923 2	68.0	2.098 2
1.0	28.625 6	35.0	6.654 3	69.0	2.035 3
2.0	27.367 9	36.0	6.397 5	70.0	1.974 7
3.0	26.166 2	37.0	6.152 3	71.0	1.916 4
4.0	25.018 4	38.0	5.918 0	72.0	1.860 3
5.0	23.922 1	39.0	5.694 2	73.0	1.806 2
6.0	22.875 2	40.0	5.480 3	74.0	1.754 0
7.0	21.875 6	41.0	5.275 9	75.0	1.703 8
8.0	20.921 2	42.0	5.080 6	76.0	1.655 4
9.0	20.010 3	43.0	4.893 8	77.0	1.608 7
10.0	19.140 8	44.0	4.715 2	78.0	1.563 7
11.0	18.311 0	45.0	4.544 4	79.0	1.520 2
12.0	17.519 1	46.0	4.381 0	80.0	1.478 2
13.0	16.763 5	47.0	4.224 6	81.0	1.437 7
14.0	16.042 6	48.0	4.075 0	82.0	1.398 6
15.0	15.354 9	49.0	3.931 7	83.0	1.360 8
16.0	14.698 7	50.0	3.794 6	84.0	1.324 3
17.0	14.072 8	51.0	3.663 3	85.0	1.289 0
18.0	13.475 7	52.0	3.537 5	86.0	1.254 9
19.0	12.906 2	53.0	3.416 9	87.0	1.221 9
20.0	12.362 9	54.0	3.301 4	88.0	1.189 9
21.0	11.844 7	55.0	3.190 7	89.0	1.159 0
22.0	11.350 4	56.0	3.084 6	90.0	1.129 1
23.0	10.878 9	57.0	2.982 7	91.0	1.100 1
24.0	10.429 1	58.0	2.885 1	92.0	1.072 1
25.0	10.000 0	59.0	2.791 3	93.0	1.044 9
26.0	9.590 7	60.0	2.701 4	94.0	1.018 6
27.0	9.200 1	61.0	2.615 0	95.0	0.993 0
28.0	8.827 5	62.0	2.532 1	96.0	0.968 3
29.0	8.472 0	63.0	2.452 4	97.0	0.944 3
30.0	8.132 7	64.0	2.375 9	98.0	0.921 0
31.0	7.808 9	65.0	2.302 3	99.0	0.898 3
32.0	7.499 9	66.0	2.231 6	100.0	0.876 4
33.0	7.204 9	67.0	2.163 6		

从表 6-2 中可以看出，10 ℃时热敏电阻的阻值只有 19.140 8 kΩ，可见小于 20 kΩ；100 ℃时热敏电阻的阻值只有 0.876 4 kΩ，也小于 20 kΩ。因此，在 10 ℃～100 ℃温度范围内，符合图 6-30 中电路 R_t < 20 kΩ 的要求。

在 0 ℃时，热敏电阻的阻值为 29.942 0 kΩ。这时，执行 A/D 转换所得的时间 T_m 有可能超出定时器计数器寄存器 TCNTR 的计数范围。

如果要扩大测量温度的范围，例如，从 0 ℃～100 ℃，那么，可以用实时中断 RTI 作为计数计时周期，而不是 TOF 中断。由于实时中断的计数是不能读出的，故还需 TOF 中断计数记录 15 级脉冲计数器的高 7 位计数。

当温度求出来之后，将其变换成 2 个显示码，存放在特定的显示单元中。

当外部$\overline{\text{IRQ}}$中断请求信号出现时，单片机响应中断，把显示码送到显示器中去显示。显示器 L_1，L_2 是进行扫描显示的。当 PA7＝0 时，送高位显示码，在 L_1 中显示高位温度；当 PA7＝1 时，送低位显示码，在 L_2 中显示低位温度。由于只有 2 位显示，所以在$\overline{\text{IRQ}}$中断中，只需对 PA7 执行求反操作，就可以控制扫描过程。这时的原则如下：

$\overline{\text{IRQ}}$中断被响应时，先取 PA7 的内容，并对 PA7 内容求反。求反之后如果有 PA7＝0，则取高位显示码，并和 PA7 内容合并之后一起送到 PA 端口；如果 PA7＝1，则取低位显示码，和 PA7 内容合并后一起送往 PA 端口。

很明显，显示器扫描显示过程相当简单。在显示器中，每个显示码只需用 7 位，每一位和一个数码管对应，显示码占用 PA 端口的低 7 位，即 PA6～PA0。故显示码和 PA7 合并时只需执行逻辑操作即可。合并之后得到的是 PA7～PA0 共 8 位的数据，送到 PA 端口就可以实现正确的显示。

在工作过程中，对 R_t 测量时的 A/D 转换是由定时器计数器寄存器 TCNTR 的溢出同步的。在具体执行过程中把 A/D 转换分 4 步来完成，即 TCNTR 溢出 4 次完成一次 A/D 转换过程。假定现在 TCNTR 处于第 k 次溢出，则有

(1) TCNTR 第 k 次溢出：进行放电，为定标做准备；

(2) TCNTR 第 $k+1$ 次溢出：进行定标计时，求出 T_s；

(3) TCNTR 第 $k+2$ 次溢出：进行放电，为检测做准备；

(4) TCNTR 第 $k+3$ 次溢出：进行检测计时，求出 T_m。

T_s，T_m 求出后，R_t 也就可以得到。之所以利用 TCNTR 溢出求 T_s，T_m，是因为在溢出的瞬间，TCNTR 的内容清"0"，而 T_s，T_m 的计时是用 TCNTR 执行的。这就意味着计时是从 0 开始的，那么在停止计时时，TCNTR 中的内容就是计时结果。这样处理十分方便。

TCNTR 溢出是一种中断，外部$\overline{\text{IRQ}}$也是一种中断。在它们同时出现时就有优先问题；另外，当一种中断正在被响应时，还存在是否允许另一种中断响应，即是否允许中断嵌套的问题。

在温度检测仪中，A/D 转换应是优先的，而显示则是次要的。相应地，TCNTR 溢出为 A/D服务，$\overline{\text{IRQ}}$中断是为显示服务，所以应该把 TCNTR 溢出放在优先地位。因此，在中断处理时，建立如下原则：

当有$\overline{\text{IRQ}}$中断响应时，允许 TCNTR 溢出中断 TOF 响应。在 TOF 响应时，不允许$\overline{\text{IRQ}}$响应。

这样，在有$\overline{\text{IRQ}}$中断响应时，一旦有 TOF 中断产生则马上响应，以保证计时的准确性。而在有 TOF 中断时，则要在 TOF 中断处理完毕之后再响应$\overline{\text{IRQ}}$中断。由于 TOF 中断的动作十

分简单，只是改变 PB0，PB1 的 I/O 形式，故而对$\overline{IRQ}$中断影响很小。但是，如果在$\overline{IRQ}$中断处理完毕，再去响应 TOF 中断，则会严重影响计时的准确性。因为，TCNTR 计时是以 μs 为单位执行计数的，而$\overline{IRQ}$中断服务需要对 PA7 求反、合并 PA7 和显示码、把合并结果送 PA 端口等动作，所用时间较多，故对 TCNTR 计时形成极大的影响，使 T_s，T_m 计时严重失真。

按钮 B 是为了出现特殊情况执行手动复位而设定的。只要一按该按钮，程序就会从头开始执行。

温度检测仪具有结构简单、元件少、经济性好的特点，在生物发酵、塑料温度测量、热水温度测量等方面都有很多用途。实际上，把电路做一些修改就可以用于检测和控制。在那些要求一般的控制场合也能发挥其应有作用。特别是它的价格低廉，很适合进行大批量生产。这时更显示出其优良的性能价格比。

6.3.4 控制软件框图

68HC05K0 控制的温度检测仪是在软件控制下工作的。控制软件由主程序、TOF 中断服务程序、$\overline{IRQ}$中断服务程序组成。它们分别用于完成不同的任务：TOF 中断服务程序用于控制 A/D 转换的充电及放电过程；$\overline{IRQ}$中断服务程序用于对显示器进行扫描显示；主程序则用来进行 A/D 转换的时间记录、热敏电阻值的求取、温度值的求取以及显示码的处理，另外，主程序还进行初始化、内存分配等工作。

用户 RAM 共有 32 个存储单元，其中最高地址的 14 个单元用于堆栈，低地址的 18 个单元用于存放一些常用或临时数据。

在用户 RAM 中为程序的正常运行设置了 10 个特定的存储单元。它们是程序运行中的专用变量单元，不允许其他临时数据占用。这些专用变量单元是程序运行中必不可少的。下面对这些特定存储单元的分配及其意义进行说明。

用户 RAM 中的 10 个特定存储单元分别是 TN，MB1，MB2，k，T，T_s，T_m，$T_f(R_t)$，DP_H、DP_L 等，它们的地址和意义如表 6-3 所示。

表 6-3 RAM 中 10 个专用单元

地址	符号	意义
$00E0	TN	TOF 中断次数计数单元，TN=0～3
$00E1	MB1	PB0 现行输入状态存储单元
$00E2	MB2	PB0 上次输入状态存储单元
$00E3	k	$1/K$ 存放单元，$K=C_6\cdot\ln[(5-V_{OL})/(5-V_{IH})]$
$00E4	T	充电时间暂存单元
$00E5	T_s	定标时间存放单元
$00E6	T_m	测量时间存放单元
$00E7	T_f,R_t	(T_m-2T_s)存放单元，R_t 存放单元
$00E8	DP_H	高位显示码存放单元
$00E9	DP_L	低位显示码存放单元

控制软件的主程序如图 6-32 所示，$\overline{IRQ}$和 TOF 中断服务程序如图 6-33 所示。由于堆栈只占了用户 RAM 高地址的 14 个单元，而 10 个专用单元只占用了 RAM 低地址的 10 个单元，所以在用户 RAM 中的中部地址处仍有 8 个存储单元可以供程序运行中暂时存放中间数据用。在程序运行时，为了进行电阻值－温度值的转换，需要用到热敏电阻的分度表。由于在 68HC05K0 中有扩展寻址方式 EXT 和 16 位偏置量变址方式 IX2，而 LDA 和 LDX 两条指令都有这两种变址方式。所以，热敏电阻的分度表可以存放在用户 ROM 里，在进行电阻值－温度值转换时，可以用 LDA 或 LDX 指令读出并和当前的电阻值比较而求取其温度值。在处理时，采用 LDA 指令并用 16 位偏置量变址方式是很容易实现这种转换的，这时可用变址寄存器 X 存放温度值，而变址寄存器 X 和偏置量相加的地址存放对应温度的电阻值。

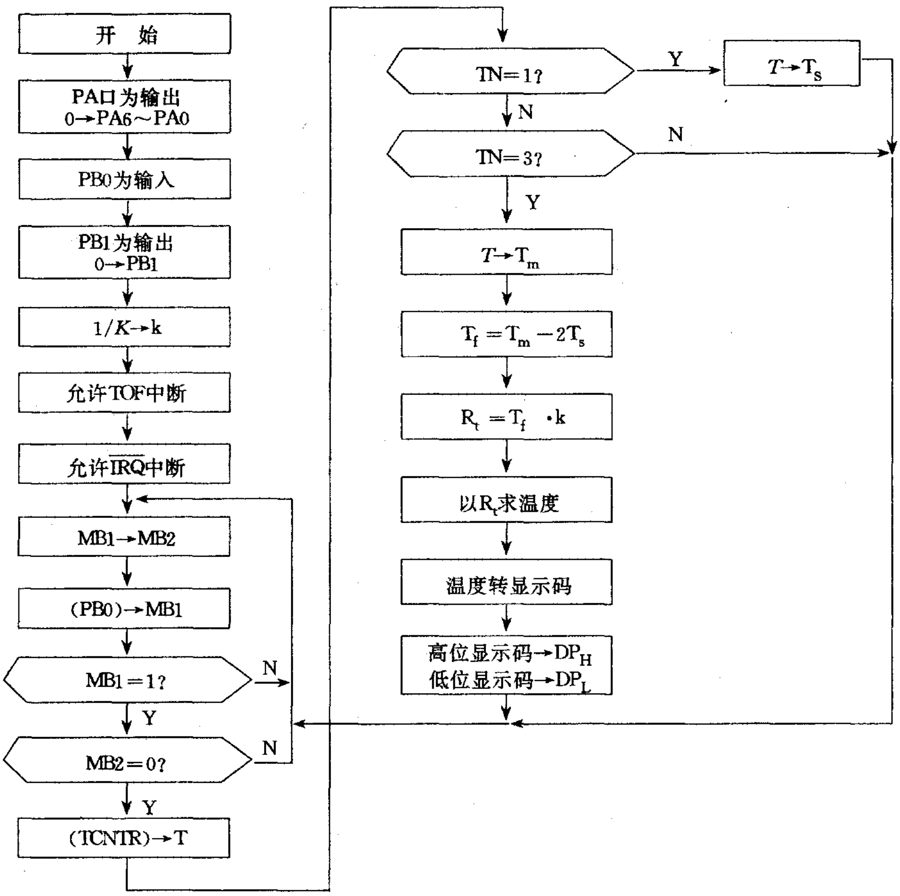


图6-32　控制软件主程序

主程序的流程如图 6-32 中所示。在复位之后程序开始执行。

首先，主程序进行初始化工作，把 PA 端口设置为输出口，准备进行扫描显示。然后，把 PB0 设置为输入口，准备测量电容 C 的电压值。同时，把 PB1 设为输出，并令其输出“0”。这时也就是对电容 C 进行放电。接着把常数 $1/K$ 送入单元 k 中。并有

$$1/K = 1.09\times10^8\ \mathrm{F}^{-1}$$

由于在式(6-60)中，T_f 是以 μs 为单位的，并且有

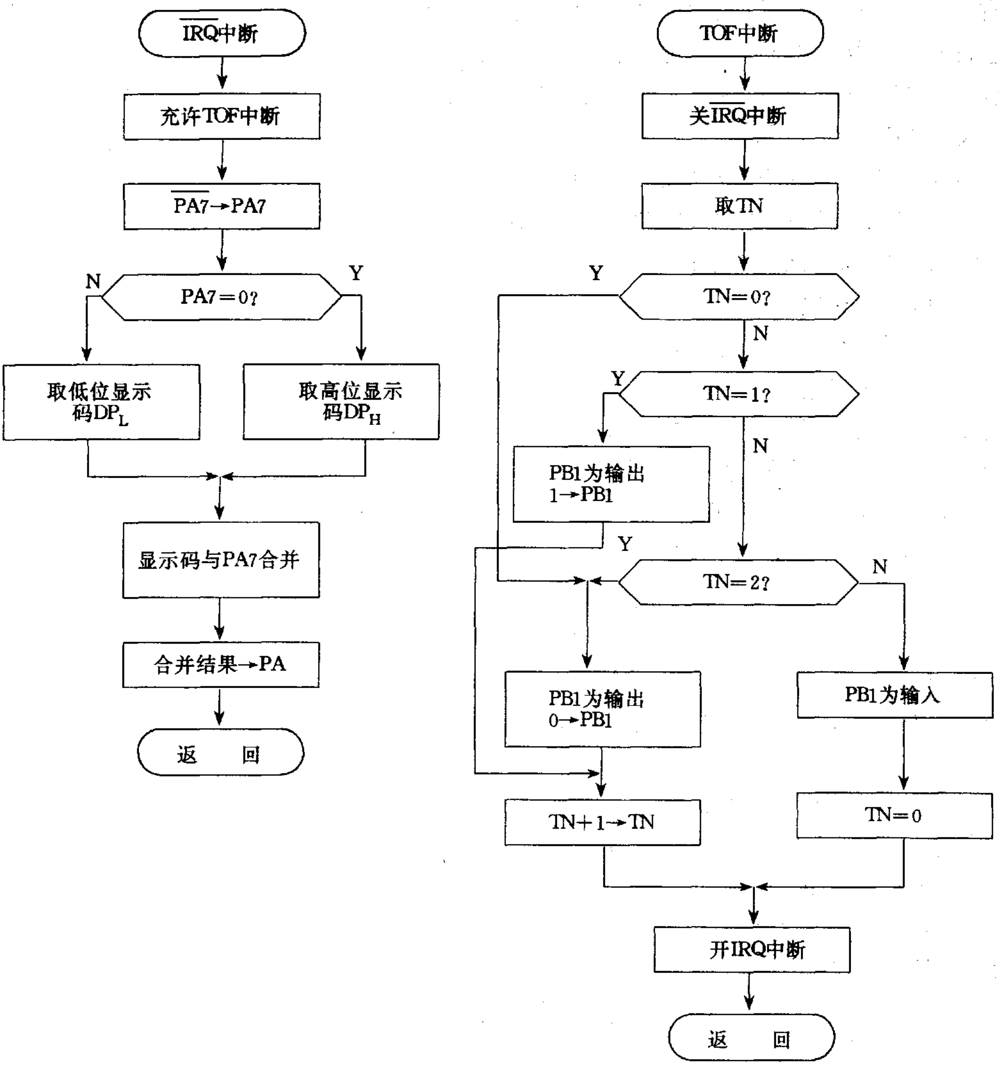


图6-33 控制软件中断服务程序

$$R_t = T_f \times 1.09 \times 10^8\ \mathrm{F}^{-1}$$

如果T_f的值以 s 为单位，则把T_f的值看成实际秒值，同时乘以系数10^{-6}，即$T_f\times10^{-6}$。这时有

$$R_t = T_f \times 10^{-6} \times 1.09 \times 10^8\ \mathrm{F}^{-1} = T_f \times 109\ \mathrm{F}^{-1} \tag{6-62}$$

式中，T_f的单位为 s。

显然，在存储单元 k 中存放值 109 即可，而在求R_t时，只要T_f的值和K相乘即可。

接着开 TOF 中断和$\overline{\mathrm{IRQ}}$中断，允许进行显示扫描和 A/D 转换。到此，初始化完毕。

在主程序初始化完毕之后，便进入工作部分的入口。

进入工作部分后，主程序首先把 PB0 上次输入状态存放在 MB2 单元中，再将 PB0 的当前输入状态存放在 MB1 单元中。

接着，对 MB1 的内容进行判断，如果 MB1＝0，说明电容C_6充电尚未达到 PB0 输入高电平门槛电压，则不用对 MB0 的内容进行判断。如果 MB1＝1，说明电容C_6充电电压达到了

PB0 输入高电平门槛电压，这时，还需要判断上次 PB0 输入的状态是否为“0”，也即对 MB2 的内容进行判断。如果 MB2=1，说明原来 PB0 的输入状态就为高电平，则不应取定时器计数器寄存器 TCNTR 的内容；如果 MB2=0，说明原来 PB0 的输入状态为低电平，也就是说 PB0 的输入状态从“0”→“1”，可见是电容 C_6 刚刚充电达到了 PB0 输入高电平的门槛电压。这时，应把 TCNTR 的内容送入充电时间暂存单元 T。

然后对 TOF 中断次数的计数单元 TN 的内容进行判断。TN 的内容所反映意义如下：

TN=0，第一次放电，为定标准备；

TN=1，定标充电；

TN=2，第二次放电，为检测准备；

TN=3，检测充电。

所以，首先判断 TN 是否为“1”，如果 TN=1，则说明刚才是定标充电，故而把充电时间暂存单元 T 的内容存入定标时间存放单元 T_s 中。如果 TN=0，则判断 TN 原来是否为 3。如果 TN 原来是 0，则说明不是检测时间，故返回程序工作部分入口。如果 TN 原来是 3，说明刚才是检测充电，故而把充电时间暂存单元 T 的内容存入检测时间存放单元 T_m。

在执行了 $T→T_m$ 之后，就可以求出 T_f。利用存于 k 单元的常数和 T_f 值相乘，就可以求出热敏电阻 R_t 值。采用对分法可以十分容易在用户 ROM 所存放的热敏电阻分度表中搜索到对应的温度值。

随后，把温度值转换成 7 段发光二极管的显示码，并把高位显示码送入 DP_H 单元，把低位显示码送入 DP_L 单元。

然后，主程序返回到工作部分的入口。

中断服务程序如图 6-33 所示。$\overline{IRQ}$中断服务程序执行扫描显示功能，定时器溢出 TOF 中断执行 A/D 控制功能。

$\overline{IRQ}$中断在电源过零时产生，故每 10 ms 产生一次中断。进入$\overline{IRQ}$中断服务程序时，首先开 TOF 中断，故在$\overline{IRQ}$中断服务程序执行时可以响应 TOF 中断。接着对 PA7 位求反，也即是对显示器 L_1，L_2 两位进行扫描，原来发光的停止发光，原来未发光的开始发光。然后判断现行发光的位，实质也是判断 PA7 是否为“0”。如果 PA7=0，说明显示高位 L_1，则取 DP_H 中的高位显示码；如果 PA7=1，说明显示低位 L_2，则取 DP_L 中的低位显示码。然后把 PA7 和显示码合并，即显示码在 PA6～PA0 低 7 位，合并结果形成 8 位数据送到 PA 端口，便实现了相应的显示。最后返回断点。

TOF 中断在定时器溢出时产生。进入 TOF 中断服务程序时，首先关$\overline{IRQ}$中断，然后把 TOF 中断计数单元 TN 的内容取出来，并进行判断。

如果 TN=0，说明应进行放电，故把 PB1 设置为输出口，并输出低电平“0”，电容 C_6 通过 R_4 和 PB1 放电。接着使 TN 加 1，然后开$\overline{IRQ}$中断后返回。

如果 TN≠0，则判断 TN 是否为“1”。如果 TN=1，说明是定标充电，把 PB1 设置为输出口，并输出高电平“1”。电容 C_6 被来自 PB1 的高电平通过 R_4 进行充电。接着对 TN 执行加 1，然后开$\overline{IRQ}$中断后返回。

如果 TN≠1，则判断 TN 是否为“2”。如果有 TN=2，说明是第二次放电，把 PB1 设置为输出口，并输出“0”，电容 C_6 通过 R_4 和 PB1 进行放电。随后对 TN 加 1，然后开$\overline{IRQ}$中断后返回。

如果 TN≠2，则说明 TN=3。这时应进行检测充电，把 PB1 设置为输入端口，则 V_{DD}即

+5 V电源过 R_t,R_2,R_3,R_4 对 C_6 进行充电。接着,把 TN 清为"0"。然后开$\overline{IRQ}$中断后返回。

TOF 中断服务程序使 TN 的值只有在 0～3 之间进行循环计数,并根据 TN 的值执行 A/D转换中的一个步骤,即充电或放电。

之所以在 TOF 溢出中断时开始充电,是因为这时定时器 TCNTR 的内容为零,此时开始充电,则 TCNTR 不断计数,直到 PB0 输入高电平门槛电压时停止,这时,TCNTR 的计数值就是充电时间。

6.4 68HC05K0 控制的微工件抛光机

68HC05K0 微工件抛光机是一种自动化抛光机器。有些工件的表面需要加工平滑,往往要进行抛光处理。对于有一定体积的工件,抛光可以用手工或机械的方法。把工件固定在一定的位置上,然后用高速旋转的软纤维对被抛光的面进行连续摩擦,使工件面被磨平磨光。对微细的工件,进行抛光是较为困难的。首先,微细工件难以固定;其次,难以构造和微细工件相匹配的高速旋转软纤维装置。为此,需要一种全新的自动化装置,可以对微工件进行全面的抛光。68HC05K0 控制的微工件抛光机就是能满足这种需要的新型机器。这种机器不但可以对微工件进行抛光,还可以对首饰以及各种小型产品进行抛光。在这一节中,将介绍 68HC05K0 对微工件抛光机的控制原理和作用。

6.4.1 微工件抛光机的基本原理

微工件抛光机如图 6-34 所示。它由抛光工件盘、磁铁、直流电动机、电机支架、电源、控制板以及抛光机壳等组成。

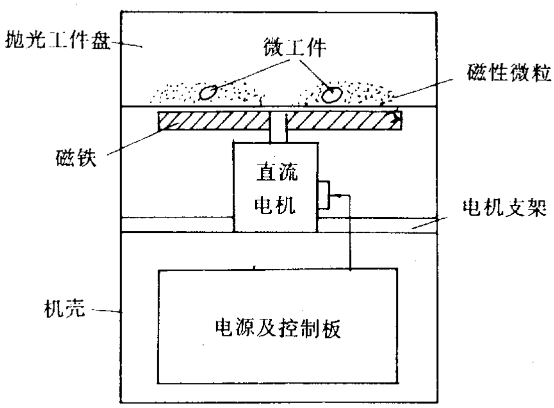


图 6-34 微工件抛光机结构

电源分两部分,一部分向直流电动机提供励磁电流,另一部分向控制板提供电源并且向直流电动机供电。

控制板是直流电动机的控制系统。它包括控制面板及控制电路印刷电路板。控制面板上要装有按钮、显示器以及调速电位器等,用于显示抛光机现行的工作状态以及供用户输入有关控制信号或设定有关工作方式。控制电路印刷电路板中安装的是以单片机 MC68HC05K0 为核心的控制电路,控制电路控制微工件抛光机的工作。它接受来自控制面板的按钮信号,从而

决定抛光机的工作方式，即进行正向抛光旋转还是反向抛光旋转，亦或是双向抛光旋转。控制电路也接收来自电位器的信号，以确定直流电动机的旋转速度，也即是抛光速度。控制电路还会把抛光机现行工作状态送到控制面板去显示，使用户知道抛光机的工作情况。

电机支架用于支撑直流电动机，使它能固定在微工件抛光机的外壳上。

直流电动机是抛光机的抛光动力部分，直流电动机旋转所产生的转动能量用于对微工件抛光。直流电机的工作方式是由控制板控制的，其工作方式有正向转动，反向转动以及双向转动三种。双向转动时，直流电动机先正向转动一定时间，然后又反向转动一定时间，随后又执行正向转动……周而复始。

磁铁固定在直流电动机的转轴上，当直流电动机旋转时，磁铁也会随之旋转。磁铁旋转的时候带动抛光工件盘中的磁性微粒旋转。由于磁性微粒像沙子一样堆在一起，当磁铁旋转时，由于磁性微粒所处的位置和状态不同，它们各自受磁铁旋转的影响不同，所以它们各自的位置变动和位移、转动速度也不同。因此，它们之间就产生摩擦，当把被抛光的微工件放入磁性微粒之中时，磁性微粒就会对工件产生摩擦作用。这种摩擦会使微工件产生抛光效果。

抛光工件盘用于放置被抛光的微工件和磁性微粒。抛光工件盘的底部和磁铁并不接触，但两者之间相距很近，所以，磁铁产生的磁力线足以通过抛光工件盘作用于磁性微粒。当直流电机旋转时，磁铁也一起旋转，则磁力线也会因之旋转，磁性微粒在旋转磁力线的作用下，各自因位势、形状的区别有着不同的旋转速度和方向。抛光工件盘中的微工件在旋转磁性微粒摩擦下被抛光。当直流电动机的转速高时，微工件被抛光的作用强烈一些；当直流电动机转速低时，抛光作用就弱一些。为了改善抛光效果，通常令直流电动机双向转动，时而正转，时而反转。这时，磁性微粒被磁力线带动反复正、反旋转，对被抛光的微工件摩擦就更加均匀，从而使抛光效果更好。

微工件抛光机有噪声小、尘埃少、抛光均匀等一系列优点，和传统软纤维抛光机有着完全不同的工作机制。

微工件抛光机的控制面板如图 6-35 所示。在面板上有 5 个发光二极管显示器，3 个用于显示速度。直流电动机的速度分为高、中、低三档。高速是指直流电动机以其标称值速度的 70%～100%工作，中速是指以标称值速度的 40%～70%工作，低速是指以标称值速度的 40%以下速度工作。面板中的调速电位器用于改变直流电动机的转动速度。当直流电动机转动时，其速度范围会在显示器 D_2～D_4 中显示出来；显示器 D_5，D_6 用于显示直流电动机的旋转方向；

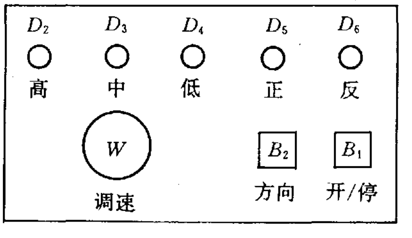


图 6-35　控制面板

按钮 B_2 用于设定直流电动机的转动方向。当按动 B_2 时，方向按正→反→双向→正的顺序循环设定。当转动方向被设定为双向时，直流电动机先正向转动一段时间，再反向转动一段时间，然后再正向转动一段时间……循环进行正、反旋转。速度范围的高、中、低显示器分别用红、绿、黄三种不同的颜色表示；正、反转显示器分别用红、绿二种不同颜色加以区别。按钮 B_1 用于启动或停止控制电路的工作。

在微工件抛光机中，抛光工件盘处于上部。上部没有盖，故而微工件可以直接投入工件盘中。在工作过程中，也可以直接从顶部观察抛光过程。磁铁安装在直流电机轴上，直流电机由

电机支架固定在抛光机中部。电源部件安装在抛光机的底部，控制板处于底部机壳内侧，而控制面板则安装在机壳下部。整个抛光机还有一个用于支撑整个抛光机的底座。由于控制面板是在机壳下部外面。故而用户可以方便地设置工作状态和了解抛光机的工作情况。

抛光机在工作过程中，磁性微粒和微工件摩擦时会产生摩擦声，但是和一般的抛光机相比，其噪声小得多；而且，所产生的粉尘也少得多。所以，这是一种对环境污染较小的抛光设备。

6.4.2 直流电动机速度控制的方法

在微工件抛光机中，直流电动机是实现抛光的动力来源。直流电动机的旋转速度和旋转方式对抛光有直接的影响。由于在抛光机中，对直流电动机的控制还采用一些特别的技术，因此需要对直流电动机的速度调节方法作一些介绍。

一、直流电动机的特性和调速机理

直流电动机有并激直流电动机、串激直流电动机和复激直流电动机等三种。在这几种电动机中，并激直流电动机的特性最硬，同时，抛光时的负载及负载变化不大。所以，在抛光机中采用并激直流电动机。

并激直流电动机指其励磁绕组是和电枢绕组并接到电源上。并激直流电机的原理如图6-36所示。

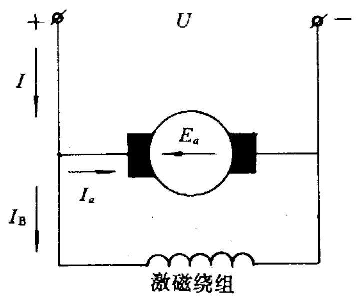


图 6-36 并激直流电动机原理

从图中看出，并激直流电动机中的电流关系如下：

$$I = I_a + I_B \tag{6-63}$$

其中，I 是电机电流；

I_a 是电机电枢电流；

I_B 是励磁电流。

当有电流流过电枢时，会产生感应电动势 E_a，感应电动势由下面公式确定：

$$E_a = \frac{pN}{60a}\Phi n \tag{6-64}$$

其中，p 是直流电动机的极对数；

N 是直流电动机总有效导线数；

a 是直流电动机绕组支路对数；

Φ 是磁通量；

n 是转速。

由于对于一个电机而言，极对数 p、总有效导线数 N、支路对数 a 是确定的常数，故而有

$$C_e = \frac{pN}{60a} \tag{6-65}$$

是常数。

从而有

$$E_a = C_e\Phi n \tag{6-66}$$

从图 6-36 中可以看出，感应电动势 E_a 是外加电压 U 和内部压降之差。

$$E_a = U - I_aR_a \tag{6-67}$$

其中，R_a 是电动机电枢内阻。

把式(6-66)和式(6-67)合并，则有

$$n=\frac{U-I_aR_a}{C_e\Phi} \tag{6-68}$$

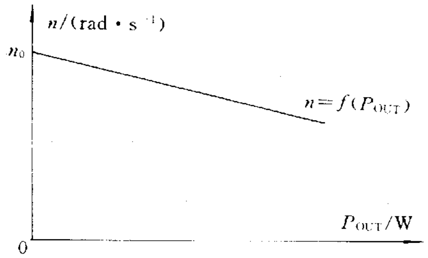


图 6-37 直流电动机的转速特性

根据式(6-68)可以得到直流电动机的转速特性。转速特性是反映转速和直流电动机输出功率 P_{OUT} 之间的关系，一般有 $P_{OUT}\approx I_aR_a$。当电压 U 为常量，励磁电流为常量，即 Φ 为常量时，转速会随输出功率的增加而稍有下降。转速特性如图 6-37 所示。

当电动机负载增加时，电流 I_a 将会增加，从式(6-68)可知，转速会下降。但由于直流电动机的电枢内阻 R_a 很小，故而转速下降不会很多。n_0 是直流电动机空载转速，当它从空载到额定负载时，转速下降 3%～8%左右。

从并激直流电动机的转速特性和式(6-68)可知，调节直流电动机转速的办法有两种：

(1) 改变电动机外加电压 U，则可以改变 $(U-I_aR_a)$ 的值，从而改变转速 n；

(2) 改变励磁电流，从而改变磁通 Φ，进而可以改变转速 n。

在实际工作中，一般是使用改变电压 U 的方法来实现对电动机调速的。当负载不变时，一般有 I_a 不变，显然，由式(6-68)可知，U 的改变会对转速有直接的控制作用。

二、晶闸管直流调压电路原理

采用晶闸管可以实现直流调压。晶闸管在直流调压中主要有两种功能：

(1) 把交流电压整流成直流电压；

(2) 以不同的导通角整流获得电压不同的直流电源。

晶闸管直流调压电路有单相、三相全控桥式电路，单相、三相半控桥式电路等。在抛光机中，由于电源是采用单相电源，因此这里介绍的是单相全控、半控桥式电路以及双向全桥电路。

1. 单相全控桥式电路

这种电路采用普通晶闸管组成一个全控桥式电路，它的电路结构如图 6-38 所示。

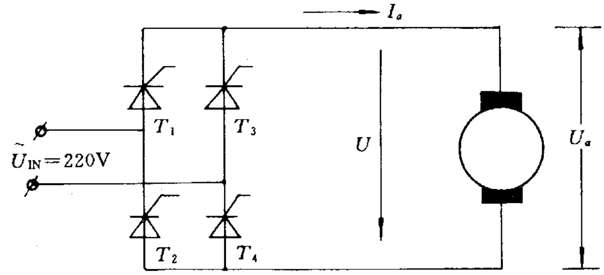


图 6-38 单相全控桥式电路

由于电路的负载是直流电动机，而直流电动机会产生反电动势 E_a，故而这是一种反电动势负载。

忽略主回路的电感，当整流输出电压 U 大于反电动势 E_a 时，才会有电流 I_a 输出。这样，晶闸管的导通角就不会是 180°。在这种反电动势负载的情况中，电压、电流波形如图 6-39 所示。

从图中可知，在晶闸管导通时，有直流电动机的端电压$U_a=U$；当晶闸管截止时，有$U_a=E_a$。图中，I_d是平均电流。

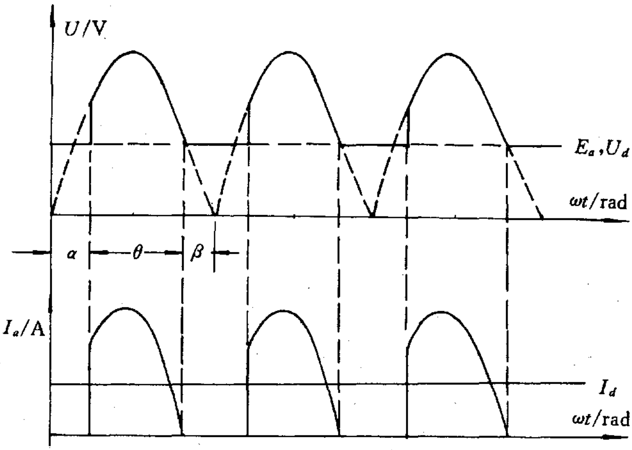


图 6-39 反电动势负载时的电路波形

由于存在反电动势E_a，故而存在停止导通角β，因为，在过零点的$\pm\beta$角之内，有$E_a\geqslant U$，晶闸管不能导通。在图 6-39 中，α是控制角，即移相角；θ是导通角；β是停止导通角。显然，电枢电流I_a只有在导通角θ期间存在。

如果电源的峰值为$\sqrt{2}U_{IN}$，反电动势E_a可以确定，则停止导通角β为

$$\beta=\arcsin\frac{E_a}{\sqrt{2}U_{IN}} \tag{6-69}$$

当$\alpha<\beta$时，为了使在控制时晶闸管能可靠触发导通，要求触发脉冲的宽度足够宽，起码要宽于$(\beta-\alpha)$，使在控制角α产生的触发作用到β角度时仍有效，从而使晶闸管导通。

如果晶闸管的整流电压平均值用U_d表示，则有

$$U_d=E_a+\frac{1}{\pi}\int_{\alpha}^{\pi-\beta}(\sqrt{2}U_{IN}\sin\omega t-E_a)d(\omega t) \tag{6-70}$$

直流电动机的平均电流I_d为

$$I_d=\frac{1}{\pi}\int_{\alpha}^{\pi-\beta}[(\sqrt{2}U_{IN}\sin\omega t-E_a)/R_a]d(\omega t) \tag{6-71}$$

而直流电动机的电流有效值为I_a，即

$$I_a=\sqrt{\frac{1}{\pi}\int_{\alpha}^{\pi-\beta}[(\sqrt{2}U_{IN}\sin\omega t-E_a)/R]^2d(\omega t)} \tag{6-72}$$

2. 单相半控桥式电路

单相半控桥式电路和全控桥式电路不同，在接成桥式的整流电路中，共有 4 个用于整流的器件，其中 2 个是二极管，2 个是晶闸管。在全控桥式电路中，用的是 4 个晶闸管，所以，它们全部是可控的；而在半控桥式电路中，只有 2 个晶闸管，所以，有一半元件是可控的。

单相半控桥式电路的结构如图 6-40 所示。

在输入电源U_{IN}为正半波时，用控制角为α的触发信号V_g对晶闸管T_3执行触发，则T_3导

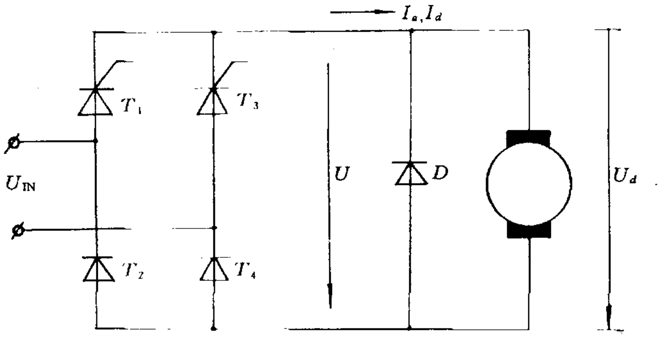


图 6-40　单相半控桥式电路

通。电源U_{IN}通过T_3,T_2对电动机供电。当电源U_{IN}变小到$U_{IN}=0$,然后再变成负半波时,由于电动机是一个电感设备,故而其中的电流会保持原有的方向,从而在U_{IN}从零到负的过程中,电动机中的电流就会过T_3,T_4续流,这也是电动机存储的电能释放的过程。

当输入电压U_{IN}处于负半波时,触发信号V_g也以控制角α对晶闸管T_1进行触发,则T_1导通。这时,电源U_{IN}通过T_1,T_4对电动机供电。当电源U_{IN}变小,再达到正半波时,电动机电流过T_1,T_2续流。

单相半控桥式电路波形如图 6-41 所示。其中,U是电压波形;I_d是平均电流,阴影部分是续流作用;V_g是触发信号。

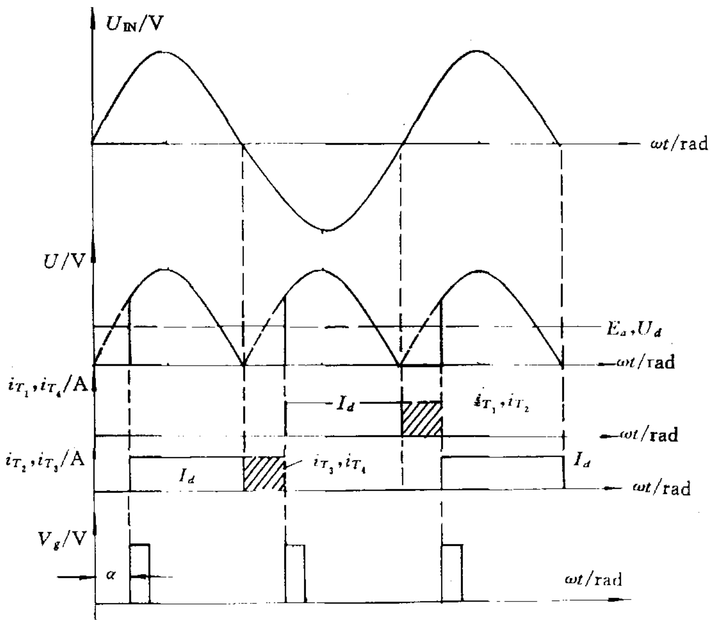


图 6-41　电路信号波形

在实际工作中,如果控制角$\alpha = 180^\circ$或突然切断触发电路,那么会发生一个晶闸管直通,同时,2个二极管轮流导通的现象,这是不正常失控现象。为了解决这种失控情况,在电动机的两

端接一个续流二极管 D。当电动机两端电压消失时，它存储的能量过 D 释放，也即过 D 续流。这样，就不再过桥式电路续流，从而保证晶闸管能够正常关断。

3. 双向全桥电路

所谓双向全桥电路，就是桥式电路全部由二极管组成，它是不可控的。而在桥式电路的交流回路中用一个双向晶闸管控制交流电压的大小。

双向全桥电路的结构如图 6-42 所示。

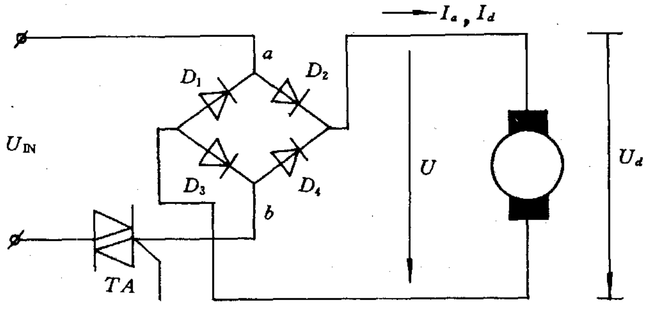


图 6-42 双向全桥电路结构

从图 6-42 中可以看出，整流桥由 4 个二极管 D_1～D_4 组成，交流回路由双向晶闸管 TA 进行电压控制。由于 TA 是双向可控的，故而在图中的 a，b 点电压 U_{ab}是交流调整电压。在电路中，平均电压 U_d 看作和反电动势 E_a 大小相同，平均电流在不同半波和时刻分别流过 D_2，D_3 或 D_1，D_4，亦或 D_3，D_4。电路的信号波形如图 6-43 所示。由于在双向晶闸管关断时，b 点没有

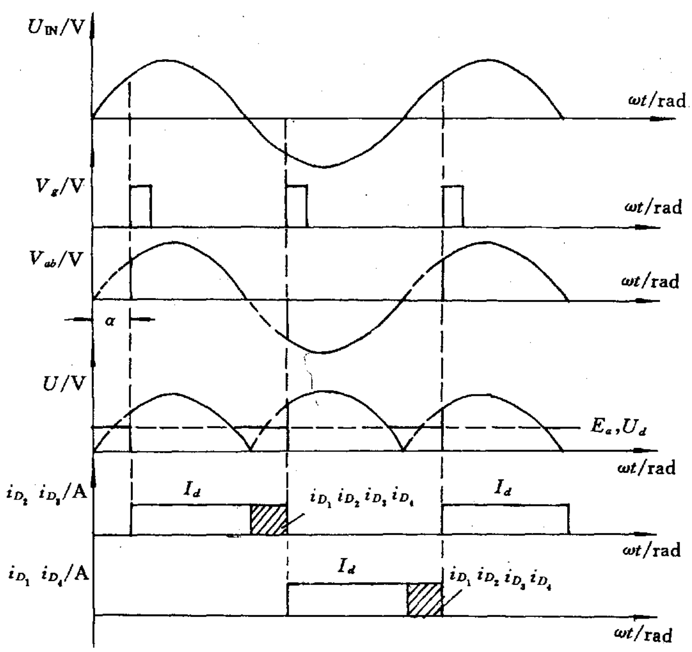


图 6-43 双向全桥电路信号波形

电压，所以，续流电流容易通过 $D_1 \sim D_4$ 这 4 个二极管续流。阴影部分的平均电流 I_d 就是通过 $D_1 \sim D_4$ 形成的。

在电源为正半波时，二极管 D_2，D_3 和晶闸管按控制角 α 触发的角度导通；在电源为负半波时，二极管 D_1，D_4 和晶闸管按控制角 α 触发的角度导通。于是形成 $D_2D_3 \rightarrow D_1D_4 \rightarrow D_2D_3$ 这样一个不断换流的过程。在控制角 α 内，晶闸管截止，这时电动机会通过有关二极管续流。虽然这时 a 点不是处于正半波就是处于负半波，但 TA 关断，a 和 b 对 U_{IN} 不能形成回路，故电动机过 $D_1 \sim D_4$ 续流。在这种电路中，显然是无需另加续流二极管的。不过，由于晶闸管 TA 的触发信号是以 b 为参考点，所以，触发信号的地应和 b 点连接。

三、晶闸管的触发方法

在微工件抛光机中，对直流电动机进行调速采用双向全桥电路。很明显，对直流电动机的调速问题就转化为对双向晶闸管进行触发的问题。

1. 双向晶闸管的 4 个触发象限

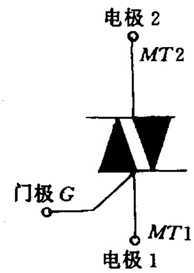


图 6-44 双向晶闸管

双向晶闸管有电极 2($MT2$)、电极 1($MT1$)和门极(G)三个电极。其中，$MT2$，$MT1$ 是主电极，G 是控制电极。在双向晶闸管导通时，电流流过 $MT2$，$MT1$；而 G 则用于施加触发信号，并控制晶闸管的开通或关断。双向晶闸管的符号如图 6-44 所示。

双向晶闸管有双向触发特性，这不但是指在外电压是正半波或负半波时可以触发，而且在门极上加上正信号或负信号时，也可以对晶闸管进行触发。

在门极上所加的触发信号及外加于 $MT2$ 的电压是对电极 1(即 $MT1$)而言的。所以，双向晶闸管有 4 个触发象限，如图 6-45 所示。

第一象限：这时加于 $MT2$ 的电压为正，门极 G 的触发信号为正，即 $MT2+$，$G+$。这时，双向晶闸管的触发灵敏度较高。

第二象限：这时加于 $MT2$ 的电压为正，但门极 G 的触发信号为负，即 $MT2+$，$G-$。这时，双向晶闸管的触发灵敏度较差。

第三象限：这时加于 $MT2$ 的电压为负，门极 G 的触发信号为负，即 $MT2-$，$G-$。这时，双向晶闸管的触发灵敏度较高。

第四象限：这时加于 $MT2$ 的电压为负，而门极 G 的触发信号为正，即 $MT2-$，$G+$。这时，双向晶闸管的触发灵敏度最差。

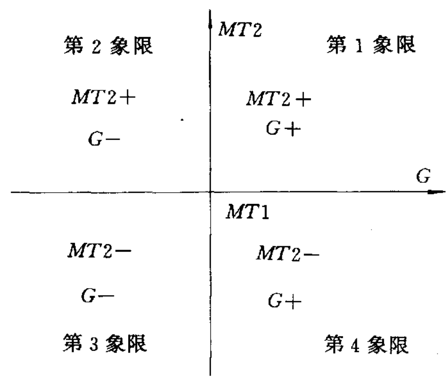


图 6-45 4 个触发象限

为了使双向晶闸管在外加于 $MT2$ 的电压正、负半波时有相近的触发灵敏度，从而有利于采用统一的触发电路并可靠触发，在实际应用中采用负信号触发，也即是第二、三象限触发。

典型的双向晶闸管的触发情况如表 6-4 所示。

2. 双向晶闸管的触发电路

双向晶闸管可以用三种基本的方法进行触发，即外加电压触发、自身电压触发和外加脉冲触发，如图 6-46 所示。其中，(a)是外加电压触发情况，(b)是自身电压触发情况，(c)是外加脉

冲触发情况。

表 6-4 双向晶闸管的触发电特性

器 件	2N6157	2N6342A	MAC15FP	MAC223
触发电流/mA				
$MT2+$ $G+$	15	6	50	20
$MT2+$ $G-$	20	6	50	20
$MT2-$ $G-$	20	10	50	20
$MT2-$ $G+$	30	25	75	30
触发电压/V				
$MT2+$ $G+$	0.8	0.9	0.9	1.1
$MT2+$ $G-$	0.7	0.9	0.9	1.1
$MT2-$ $G-$	0.85	1.1	1.1	1.1
$MT2-$ $G+$	1.1	1.4	1.4	1.3
导通电流/A	30	12	15	25

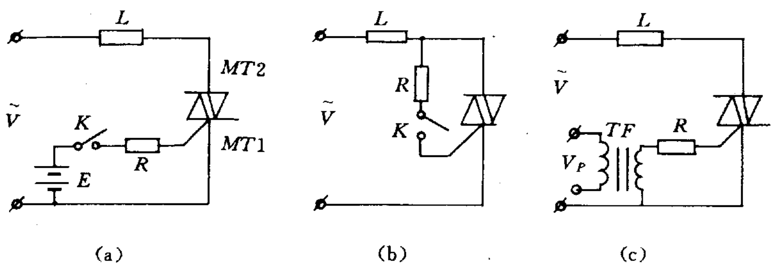


图 6-46 基本触发方法

在图 6-46 中，L 是负载，$\tilde{V}$ 是交流电源，R 是触发电源限流电阻。

在图 6-46(a)的外加电压触发情况中，当开关 K 闭合时，外电源 E 通过电阻 R 加到双向晶闸管的门极，只要电流足够大，就可以触发双向晶闸管导通。

在图 6-46(b)的自身电压触发中，当开关 K 闭合时，在双向晶闸管的 $MT2$ 极中所加的交流电压就会通过电阻 R 加到其门极，只要 R 选值恰当，就可以使双向晶闸管在 $MT2$ 电压的绝对值达到某一值而触发导通。

图 6-46(c)所示是外加脉冲触发的情况。外加脉冲 V_p 通过变压器 TF 后交连到副边，从而触发双向晶闸管。当然，这时对电阻 R 也有一定的阻值要求。

实际应用中，采用逻辑电路作为开关，控制对双向晶闸管的触发。控制电路有各种各样的结构。典型的电路有如下若干种。

外加正电压触发控制电路

外加正电压触发控制电路如图 6-47 所示。当反相器 F 输入为正脉冲时，其输出为“0”，从而使晶体管 T 截止。这样，+5 V 电源便加到晶闸管 TA 的门极，电流过电阻 R 也加到门极，使 TA 导通。当反相器输入为低电平时，晶体管 T 导通，晶闸管 TA 的 G 极和 $MT1$ 处于同电位，所以 TA 不会导通。很明显，晶体管 T 的开关状态可以控制 TA 的导通或截止。

外加负电压触发控制电路

外加负电压触发控制电路如图 6-48 所示。从图中可看出，双向晶闸管的 $MT1$ 极和+5 V 相连，而 G 极通过电阻 R、晶体管 T 和地相连；也即是说，G 极在 T 导通时处于地电平。显然，相对于 $MT1$ 极而言，G 极的电平为负，故而，这是负电压触发控制电路。如果双向晶闸管 TA 的 $MT1$ 极和地相连，那么，只要把晶体管 T 的发射极和−5 V 相连即可。

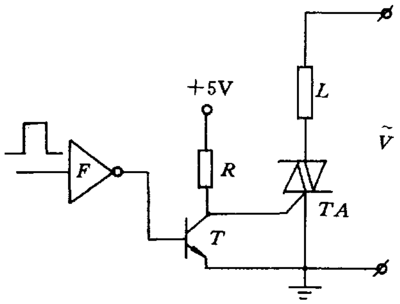


图 6-47　外加正电压触发

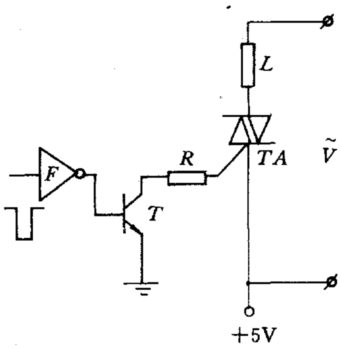


图 6-48　外加负电压触发

当外加控制信号为负脉冲时，反相器 F 输出正脉冲，正脉冲加到晶体管 T 的基极，令晶体管 T 导通，于是负电压加到 TA 的 G 极令其导通。所谓负电压是指相对于 $MT1$ 极的电平而言的。电阻 R 用于限制流过 G 极的电流，以免过大。

自身电压触发控制电路

自身电压触发控制电路如图 6-49 所示。这种电路一般采用光电耦合管进行高、低压隔离，以避免控制的开关电路承受主电路中的高压。

自身电压触发电路由光电耦合管 OC 和电阻 R 组成。当有正脉冲加入时，光电耦合管中的发光二极管导通并发光，使其光敏管导通，则加在双向晶闸管 TA 上的电压也会过电阻 R 加到其 G 极上，从而触发 TA 导通。

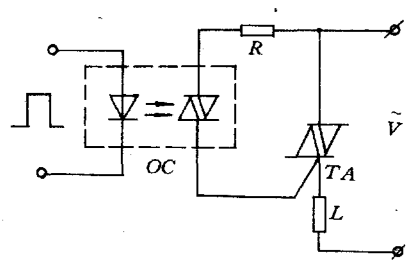


图 6-49　自身电压触发

这种触发电路的优点在于，低压部分的逻辑电路和高压部分的双向晶闸管回路绝缘。因此，在安全和干扰隔离方面有着明显的优点。但是，由于光电耦合管较贵，在经济性上来说不很合算。

外加脉冲触发电路

外加脉冲触发电路如图 6-50 所示。这种电路的特点是依靠脉冲变压器来交连外加脉冲。

在图 6-50 中，脉冲发生器由整流稳压电路和单结晶体管 CJ 组成。其中，桥式整流器 D_1，电阻 R_1 和稳压管 D_2 组成了整流稳压环节。D_1 把交流电压整流成全波脉冲直流电压，通过 R_1 限流之后由稳压管 D_2 稳压。D_2 稳压在 8 V，并作为单结晶体管 CJ 回路的电源。

电阻 R_2，R_3，电容 C 和单结晶体管 CJ 组成了脉冲发生电路。稳压电源过 R_2 对电容 C 充电，当电容充电达到单结晶体管导通门槛电平时，单结晶体管 CJ 导通，电容 C 也随之过 CJ 放电。这样，在脉冲变压器 PT 的初级绕组就产生一个正脉冲。当电容 C 放电使其电压下降到截止门槛电平时，单结晶体管截止，正脉冲结束。随后，电源又过 R_2 对 C 充电，周而复始，则不断

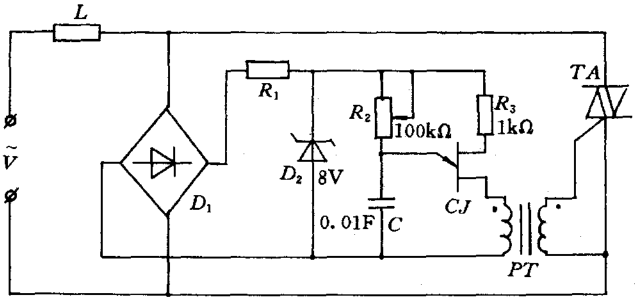


图 6-50　外加脉冲触发

产生脉冲信号。改变 R_2 的大小可改变脉冲产生频率。

脉冲信号通过脉冲变压器 *PT* 交连到双向晶闸管的 *G* 极和 *MT*1 极之间，从而触发 *TA* 导通。在单片机控制中，是无需单结晶体管脉冲发生电路的，单片机输出的脉冲信号可以通过变压器交连到双向晶闸管的 *G* 极并触发其导通。

6.4.3　控制电路及其工作原理

微工件抛光机的控制电路如图 6-51 所示。它是以单片机 68HC05K0 为核心的直流电机双

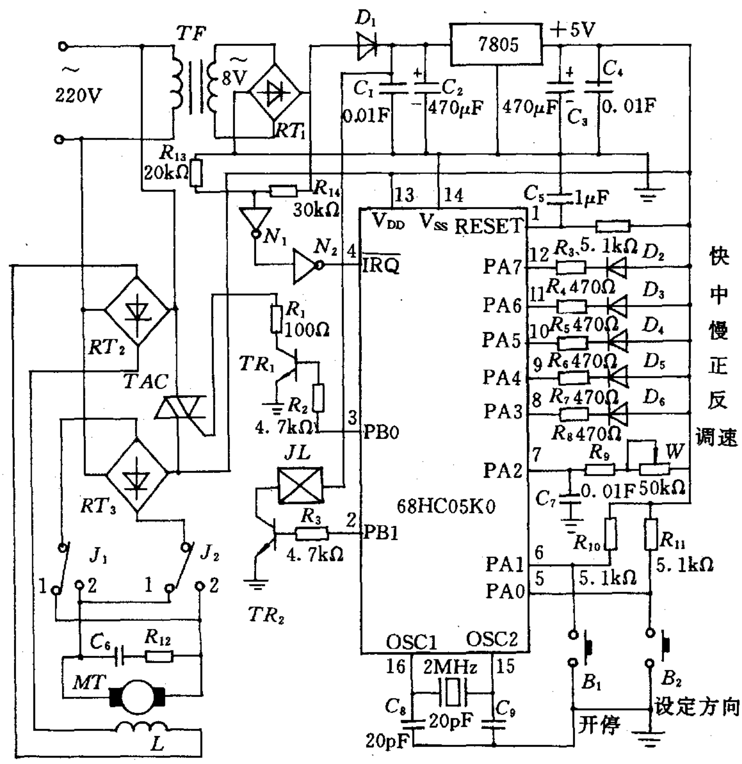


图 6-51　控制电路原理

向速度控制电路，由整流稳压电路、输入电路、显示电路、单片机 68HC05K0 以及直流电机控制回路组成。

一、整流稳压电路

整流稳压电路由整流电路和稳压电路组成。

整流电路包括变压器 TF 和桥式整流器 RT_1。变压器 TF 把 220 V 的交流电压降压成 8 V 交流电压，以便整流之后形成低压稳压电源。桥式整流器 RT_1 用于全波整流，以获得全波脉冲直流电压。该脉冲直流电压是低压的，它有两个用途：未滤波的脉冲直流电压用于产生过零脉冲信号，而脉冲直流电压经滤波之后用于形成低压稳压电源。脉冲直流电压和滤波直流电压之间用二极管 D_1 隔离。

稳压电路由电容 C_1,C_2,C_3,C_4 和稳压集成电路 7805 组成。电容 C_1,C_2 对脉冲直流电压进行滤波，从而得到较为平滑的不稳定电压，该电压有较大的波纹。然后，由稳压集成电路 7805 进行稳压控制，输出稳定的 5 V 直流电压。电容 C_3,C_4 用于抗干扰以及提高负载能力。C_4 的容量较小，它可以把高频干扰信号旁路，从而起到抗干扰作用。C_3 是容量较大的电容，它可以在负载情况较重的情况下提供部分电能，从而使稳压集成电路 7805 在过载的瞬间能够保持正常调节功能，使电源稳定在＋5 V 。

二、输入电路

输入电路包括按键输入电路、调速输入电路和过零脉冲输入电路。

1．按键输入电路

按键输入电路有启/停按键输入，设定按键输入两个。它们的电路结构相同，分别由一个电阻和一个按键串联组成。

电阻 R_{10}和按键 B_1 组成了启/停输入电路。它用于启动或停止抛光机的工作。按 B_1 一次处于一种状态，再按 B_1 一次则转为另一种状态；也即是在启动和停止这两种状态间进行选择。

电阻 R_{11}和按键 B_2 组成设定输入电路，用于设定电动机的旋转方向。按 B_2 一次它处于一种状态，再按时，则自动进入下一状态。B_2 的状态转换有三种：正转、反转、双向转。按 B_2 可以循环选择这三种状态之一。双向转动时，电动机正向转一定的时间，然后停若干秒钟，接着又反向转一定的时间，然后又停若干秒，又正向转一定的时间……不断周而复始执行。

按键输入电路工作是在软件控制及配合下完成的。

2．调速输入电路

调速输入电路由电阻 R_9、电容 C_7 和电位器 W 组成。

在工作时，PA2 口首先要输出低电平，电容 C_7 通过 PA2 放电。接着，PA2 变成输入口，电容 C_7 过电位器 W、电阻 R_9 充电。当电容电压到达 PA2 的输入高电平门槛电压时，表示电容 C_7 的电压为逻辑“1”。单片机内部对电容 C_7 开始充电到电压为“1”的充电时间，反映了电位器 W 值的大小；也即实现了时间—电阻转换，从而可以求出电位器 W 的即时电阻值。该值用于表示对直流电机的调速给定值。由于电位器 W 的值是可以改变的，因此对直流电机是可以调速的。单片机直接依据电容 C_7 充电到“1”的时间长短对直流电机速度进行调整。这样，电位器 W 的值越小，直流电机的速度越高，反亦反之。

3．过零脉冲输入电路

过零脉冲输入电路由电阻 R_{13}，R_{14}和反相器 N_1，N_2 组成。

由于桥式整流电路 RT_1 输出的脉动直流电压有较高的峰值，对反相器 N_1，N_2 造成危险，

故由电阻 R_{13},R_{14}进行分压,然后送入 N_1 的输入端。脉动直流电压经由 N_1,N_2 两级反相器整形之后,形成前后沿很陡的负脉冲,该负脉冲和电源的过零点对应,故而称该负脉冲为过零脉冲。过零脉冲加到单片机 68HC05K0 的中断输入端$\overline{\mathrm{IRQ}}$,产生中断请求。过零脉冲中断用于对双向晶闸管的触发同步。

三、显示电路

显示电路由发光二极管 D_2～D_6 和电阻 R_4～R_8 组成,用于显示抛光机现行的工作状态和工作方式。

工作方式有正转、反转、双向转三种。当正转时,正转指示发光二极管亮;反转时,反转指示发光二极管亮;正、反转时,正、反转指示发光二极管同时亮。正转指示由 D_5 和 R_7 组成,反转指示由 D_6 和 R_8 组成。

工作状态有快速、中速、慢速三种。工作状态反映的是抛光机直流电机的旋转速度。快速由 D_2 和 R_4 组成的显示回路显示,中速由 D_3 和 R_5 组成的电路显示,慢速则由 D_4 和 R_6 显示。在速度显示时,某一时刻只能显示一种状态,即只有一个发光二极管显示。

由于 PA 端口的最大灌入电流为 8 mA,所以,显示回路的电阻值不能太小。恰当选择回路中的限流电阻才能保证正确显示。电阻太大,则发光二极管不亮;电阻太小,则电流会太大而损坏单片机的 PA 口。一般取显示电流为 5 mA～8 mA,因此,显示回路的电阻选择如下:

$$\frac{V_{\mathrm{DD}}-V_D}{8\ \mathrm{mA}} \leqslant R \leqslant \frac{V_{\mathrm{DD}}-V_D}{5\ \mathrm{mA}} \tag{6-73}$$

其中,V_{DD}是电源电压,$V_{\mathrm{DD}}=5$ V;

V_D 是发光二极管压降,$V_D\approx1.3$ V。

于是有

$$462\ \Omega \leqslant R \leqslant 740\ \Omega$$

考虑到发光二极管需要有一定的亮度,故在实际中取标准电阻值 $R=470\ \Omega$。这时,流过发光二极管的电流约为 7.8 mA。在这个电流级别,发光二极管的亮度已足够了。

四、单片机 68HC05K0

单片机 68HC05K0 是一种廉价单片机,引脚只有 16 只。在抛光机控制中,它内部的部件最关键的有输入端口、输出端口、定时器和中断系统。

1. 输入端口

在抛光机控制中,输入端口用于两种不同的端口操作。一种是电平输入端口,另一种是 A/D转换端口。

电平输入端口为 PA0,PA1 端口。它们的作用是接收按键的输入信号。当按键按下时,对应端口输入低电平;按键释放时,对应端口输入高电平。程序对 PA0,PA1 的输入信号进行检测,可以很容易地判断按键是否按下,从而对工作方式进行设定以及对控制直流电动机的情况进行启动或停止。

A/D 转换端口为 PA2。它主要是把电位器 W 的现行电阻值转换成数字量。这个过程是在程序控制下执行的。A/D 转换过程如下:

(1) PA2 端口为输出口,并且输出低电平;这时,电容 C_7 过 PA2 端口放电;电容 C_7 的电压为低电平。

(2) PA2 端口为输入口,+5 V 电源过电位器 W 和电阻 R_9 对电容 C_7 充电;单片机

68HC05K0 对充电过程进行计数计时；

(3) 当电容 C_7 充电到达 PA2 输入高电平门槛电平时，单片机停止计时，这时所计得的数字和电位器 W 的阻值对应。

设电容 C_7 的充电时间用 T 表示，一般有

$$T=(W+R_9)\cdot C_7=W\cdot C_7+R_9\cdot C_7 \tag{6-74}$$

由于电容 C_7、电阻 R_9 为常量，故而令

$$K=R_9\cdot C_7$$

有

$$T=W\cdot C_7+K \tag{6-75}$$

即有

$$W=\frac{T-K}{C_7} \tag{6-76}$$

在式(6-76)中，K 和 C_7 为常量。很容易看出，电位器 W 的值和充电时间 T 有关并对应。

2. 输出端口

输出端口有两种，一种是显示端口，另一种是控制端口。

显示端口由 PA3～PA7 组成。在任一位端口输出高电平时，对应发光二极管不亮；输出低电平时，对应发光二极管发亮。

控制端口由 PB0，PB1 组成。它们用于控制直流电动机的旋转速度和旋转方向。PB1 端口是旋转方向控制端口，当 PB1 输出为低电平时，直流电动机正向旋转；当 PB1 输出为高电平时，直流电动机反向旋转。PB0 端口是速度控制端口。在过零脉冲同步下，以过零脉冲为基准，延时一段时间在 PB0 输出高电平，则可以使双向晶闸管导通。延时越短，则双向晶闸管导通角越大，输出电压越高；延时越长，双向晶闸管导通角越小，输出电压越低。输出电压越高，直流电动机转速越高，反亦反之。在 PB0 输出低电平，则直流电动机停止转。因为这时双向晶闸管不导通。在 PB0 输出的正脉冲是触发双向晶闸管的触发信号，该脉冲一般有一定的宽度。在实际应用中，脉冲宽度随导通角的大小而变化。导通角越大，脉冲宽度应越大，反亦反之。PB0 端口的控制情况如图 6-52 所示。

3. 定时器

定时器主要用于 A/D 转换和双向晶闸管的移相控制。

A/D 转换是利用定时器计数器寄存器 TCNTR 的溢出中断及其计数内容来执行的。

由于 $(W+R_9)\cdot C_7$ 的结果小于 1 ms，也即是电容 C_7 充电或放电都小于 1 ms。在 A/D 转换时定时器的作用如下：

(1) TCNTR 溢出时，开始放电，则在下一次溢出时 C_7 会放电完毕；

(2) 在 C_7 放电完毕后，TCNTR 溢出，则可以开始对 C_7 充电。在 C_7 充电达到 PA2 的输入高电平门槛电压时，把 TCNTR 的内容取出。该计数值即是 C_7 充电时间。这个时间对应于电位器 W 的值。

在双向晶闸管移相控制时，利用 TCNTR 的溢出中断及其溢出次数来执行。双向晶闸管移相控制分 9 级。控制角 α 的大小用延时来表示。这 10 个级别对应于过零脉冲分别延时 0 ms ～8 ms。

当过零脉冲请求中断时，一旦中断得到响应，就由 TCNTR 的溢出中断次数进行计时，利用 TCNTR 溢出次数进行延时。当外部振荡器频率为 2 MHz 时，TCNTR 的溢出计数周期为

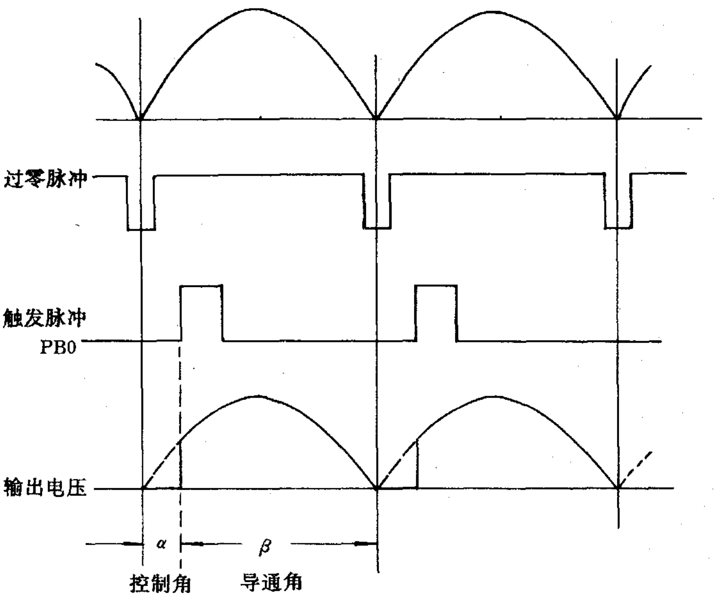


图 6-52　PB0 端口的控制作用

1 ms。因此，很容易形成 0 ms～8 ms 的 9 级延时，每一级的级差为 1 ms。故而可产生延时为 0 ms，1 ms，2 ms，…，8 ms 的控制角 α。为了使触发信号在延时不同时宽度不同，可以确定延时和触发脉冲宽度的总时间为 9.0 ms，即有 $T_{sum}=9.0$ ms。延时时间（即控制角）用 T_d 表示，触发脉冲宽度用 T_p 表示，则有

$$T_{sum} = T_d + T_p = 9.0 \text{ ms}$$

很明显

$$T_p = 9.0 \text{ ms} - T_d \tag{6-77}$$

式(6-77)中，从过零脉冲产生中断开始，TCNTR 溢出 9 次就会对应 9 ms 时间。在内存中设置一个单元，用于累加 TCNTR 过零中断后的溢出次数，并用 TN 表示。当 TN 的内容等于 T_d 时，产生触发脉冲；当 TN 的内容等于 9 ms 时，触发脉冲结束。这样，不同的延时就对应了不同的触发脉冲宽度。

4. 中断系统

在抛光机中需要两种中断：过零中断和定时器计数器寄存器 TCNTR 溢出中断。

过零中断由外部过零脉冲从 $\overline{IRQ}$ 端请求产生。因此，每当电源过零时，便产生过零中断请求。

TCNTR 溢出中断是由定时器产生的。由于振荡器的频率为 2 MHz，所以，TCNTR 的溢出周期为 1 ms。

在中断系统中，过零中断处理过程中允许 TCNTR 溢出中断，从而能进行正常延时控制以及触发脉冲宽度控制。

5. 直流电机控制回路

直流电机控制回路由桥式整流器 RT_2、RT_3，双向晶闸管 TAC，继电器 JL，晶体管 TR_1、TR_2，直流电机 MT 和有关电阻、电容组成。

桥式整流器 RT_1 用于产生直流的励磁电流。所以，这是一个他激式的直流电动机。专门的励磁回路保证了励磁的稳定性，有利于对直流电机进行调速。

桥式整流器 RT_3 用于向直流电机的电枢回路提供直流电压。双向晶闸管 TAC 用于控制直流电压的大小。

继电器 JL 用于控制直流电机的正、反转。当继电器 JL 没有吸合时，触头和 1 端接触，直流电机正转；当继电器通电时则吸合，触头和 2 端接触，直流电机反转。

在 PB1 输出低电平时，继电器 JL 不通电，故继电器不能吸合，这时电机正转；反之，PB1 输出高电平时，继电器 JL 通电，故 JL 吸合，这时电机反转。

在 PB0 输出的是 TAC 的触发脉冲。PB0 输出低电平时，双向晶闸管 TAC 截止，故直流电机不转。在过零脉冲的同步下，PB0 输出延时的正脉冲，从而触发 TAC 实现调压，最后对直流电机进行调速。双向晶闸管 TAC 的 $MT1$ 极和 +5 V 电源相连，所以，TAC 是采用负脉冲触发的。在 PB0 输出正脉冲时，过晶体管 TR_1 反相之后形成负脉冲，再去触发 TAC。显然，双向晶闸管 TAC 工作于第二、三象限，即采用 $MT2+$，$G-$ 或 $MT2-$，$G-$ 的触发方式。

6.4.4 抛光机的控制软件框图

抛光机控制软件由主程序和中断程序组成。主程序的功能是对单片机 68HC05K0 的有关寄存器、I/O 端口、内存单元进行初始化，对按键进行状态检测以及对工作方式及工作状态进行显示。

中断程序分为过零中断和定时器溢出中断。过零中断用于双向晶闸管移相控制时的过零点同步。定时器溢出中断用于控制 A/D 转换时的放电及充电过程，取得 A/D 转换结果和产生双向晶闸管移相的控制脉冲。

在控制软件工作时，需要在存储器中占用一些专门的存储单元，用以存放特殊的数据信息。这些数据信息的表示符号及其意义如下：

TN——定时器溢出次数累加计数单元。

DT——直流电动机旋转方向标志。DT＝0，表示直流电动机正向转动；DT＝1，表示直流电动机反向转动。

SP——直流电动机旋转速度标志。SP＝0，表示速度为快速；SP＝1，表示速度为中速；SP＝2，表示速度为慢速。

AD——A/D 转换状态标志。AD＝0，表示对电容 C_7 放电；AD＝1，表示对电容 C_7 充电。

ADT——电容 C_7 充电时间存储单元。

DL——和 ADT 对应的双向晶闸管移相延时时间存放单元。移相在抛光机中不用角度表示，而用时间表示。DL 存放的是移相时间，单位是 ms。

ST——启停标志。ST＝1，表示启动。

一、主程序的执行过程

主程序的流程如图 6-53 所示。

在复位或接通电源时，便进入主程序。

主程序首部是初始化部分。首先把 PA0～PA2 设置为输入端口，准备接收按键 B_1，B_2 的输入状态和进行 A/D 转换。接着把 PA3～PA7 设置为输出端口，准备显示抛光机的直流电动机的工作方式和工作状态，即转动方向和转动速度。紧接着把高电平送出到 PA3，PA5，PA6

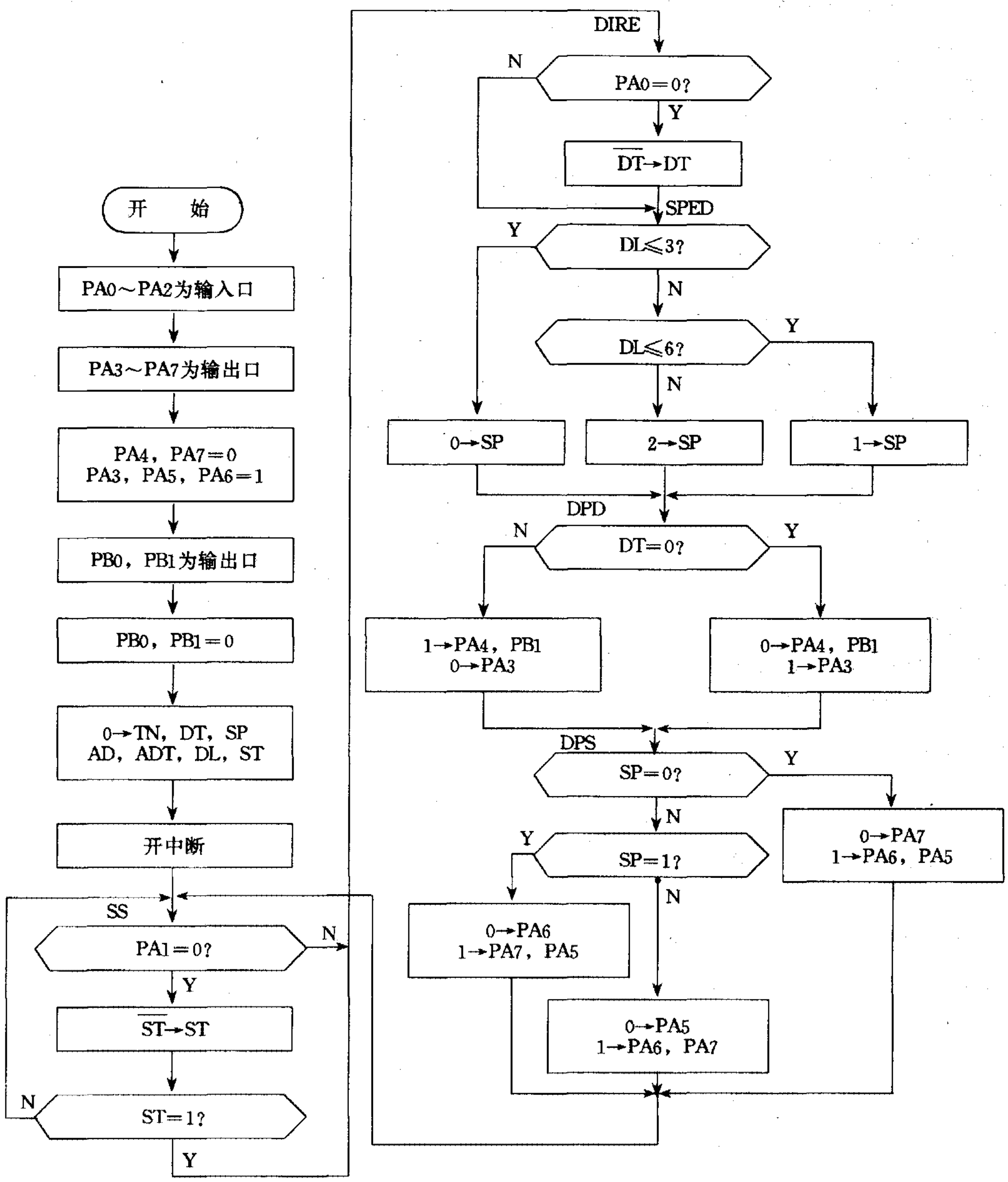


图6-53　主程序流程

端口，把低电平送出到PA4，PA7端口，从而使正向转动指示显示器和高速旋转指示显示器发光。随后，把PB0，PB1设置为输出端口，准备对直流电动机进行控制；把低电平送到PB0，PB1，使直流电动机处于正转方式，并处在停止状态。最后，对特殊存储单元清“0”，即把“0”送入TN，DT，SP，AD，ADT，ST和DL存储单元，表示定时器溢出次数为0，直流电动机的旋转方向为正向，旋转速度为快速，电容C_7处于放电状态，电容C_7充电时间存放单元清“0”，移相时间为零，处于停止机状态。开中断，允许过零中断和定时器中断响应。

SS是启停程序段，用于启动或停止程序的执行。开始时，检测启/停按键B_1的状态。如果

PA1=0，表明按键 B_1 按下，则改变抛光机原有状态，对启/停标志 ST 求反。然后判断 ST 是否为“1”，如果不是，说明停机，故而转回 SS 入口处；如果 ST=1，说明开机，程序向下执行。

DIRE 是主程序的一个程序段，用于检测方向设置按键 B_2 的状态。如果 PA0=0，说明按键 B_2 按下，则对旋转方向标志 DT 求反，改变旋转方向；如果 PA0=1，说明按键 B_2 未按下，故不对 DT 求反，保持原有旋转方向。

SPED 是速度判断程序段。在该程序段中，对移相延时 DL 进行判断，如果 DL≤3 ms，说明直流电动机处于快速旋转状态，把“0”送入速度标志 SP；如果 3 ms<DL≤6 ms，则处于中速旋转状态，把“1”送入 SP；如果 DL>6 ms，则处于慢速旋转状态，则把“2”送入 SP。

DPD 是方向显示程序段。这时，对方向标志 DT 进行判断，DT=0，说明正向旋转直流电机，所以，把“0”送 PA4，PB1 端口，令正向旋转指示二级管发光，同时使直流电机处于正向旋转方向；同时，把“1”送 PA3 端口，使反向旋转指示二极管熄灭。如果 DT=1，说明是反向转动，把“1”送 PA4，PB1 端口，把“0”送 PA3 端口，从而令反向旋转指示发光二极管亮，正向发光二极管不亮；同时，使直流电动机反向旋转。

DPS 程序段是速度显示程序段。SP=0 时，把“0”送 PA7 端口，快速显示管亮；“1”送 PA6，PA5 端口，使中、慢速显示管不亮。SP=1 时，是中速旋转，把“0”送 PA6 端口，“1”送 PA7，PA5 端口，则只有中速显示管亮。SP=2 时，是慢速旋转，把“0”送 PA6 端口，把“1”送 PA7，PA6 端口，使低速显示管发光。

DPS 程序段执行完毕则返回 SS 程序段入口。

二、中断程序的执行过程

中断程序包括过零中断处理程序和定时器溢出中断处理程序，其框图如图 6-54 所示。

过零中断处理程序十分简单，只是在过零脉冲请求中断时，把定时器溢出中断次数累加计数单元 TN 清“0”，为双向晶闸管移相触发作准备。

定时器溢出中断处理程序所执行的工作稍多一些。它有三个任务：第一是产生双向晶闸管的导通触发脉冲，第二是对 A/D 过程中的放电过程进行控制，第三是对 A/D 过程中的充电过程进行控制并求出对应的移相延时。

在定时器溢出中断处理程序的首部，首先开过零中断。接着，PG 程序段产生触发双向晶闸管的脉冲。触发脉冲分为两部分，即以过零点为基准参考点的延时和触发脉冲的宽度。首先判断 TN 是否等于 DL，也即是延时是否足够。如果 TN=DL，说明延时已够，亦即移相已足够，故马上把“1”送到 PB1 端口，触发晶闸管导通。如果 TN≠DL，则判断 TN 是否等于 9，即已达 9 ms，如果有 TN=9，说明触发脉冲应结束，故而把“0”送到 PB1，关断晶闸管。如果 TN≠9，则保持 PB1 原有状态不变。随后，使 TN 的内容加 1。接着对 A/D 转换状态标志 AD 进行判断。

如果 AD=0，说明现行状态是放电，则进入 DH 程序段。这时，把 PA2 设置为输出端口，并输出 0 电平，使电容 C_7 过 PA2 口进行放电。由于电容 C_7 可在一个溢出中断周期内放电完毕，所以下一次溢出中断时可进行充电。因此，接着把 AD 置为“1”。

如果 AD=1，说明当前状态是充电，则进入 CH 程序段。这时，把 PA2 端口设置为输入，开始进行充电计时。由于充电同样在一个溢出中断周期之内可完毕，故把 AD 置为“0”。接着，判断 PA2 口的输入是否为高电平门槛电压，若不是则一直等待；如果是则取定时器计数器寄存器 TCNTR 的内容，并送入充电时间存放单元 ADT。然后，把这个时间值转换成移相延时并存

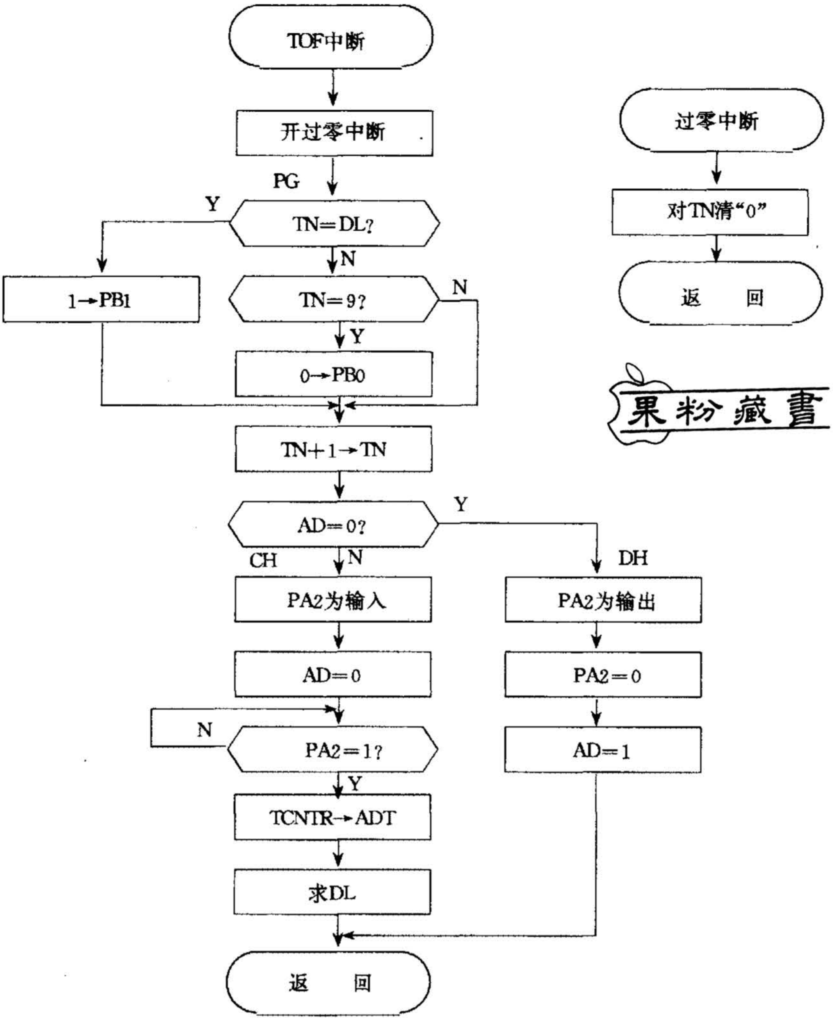


图6-54　中断程序流程框图

果粉藏書

入 DL 中。移相时间存放单元 DL 的值在主程序中用于判断速度的档次，也在溢出中断处理程序中用于产生触发脉冲。

封面设计：浩 峰

制版印刷：北京地大彩印厂

ISBN 7-81012-764-0

9 787810 127646 >

ISBN 7-81012-764-0/TP · 272

定价：19.50 元